蔬菜类商品自动定价与补货决策优化问题

摘要

本文主要建立在关联性算法、时序分析模型、需求弹性模型寻求商品供需关系及其各品类及单品之间的相互关系，运用优化算法、启发式算法对蔬菜类商品自动定价与补货决策优化问题。

首先对附件进行合并，发现销量存在明显的季节性变化，且对各品类与单品在时间序列下都有较为平滑且规律的变化，但存在部分特殊时间点出现销量异常的情况。在异常点过后恢复正常。将这类特殊时间点剔除不纳入后续的研究与分析。

针对问题一，绘制饼图、箱型图等寻求各品类的销量分布，发现花叶类存在较大的销量，占总销量的 42.2%，且花叶类存在较多异常值。对于各个单品之间的分布采用绘制直方图的方式进行观察，发现芜湖青椒（1）、与西兰花有着较高的销售量。对于品类及单品之间的相互关系的探寻，采用皮尔逊系数进行初步探寻，发现辣椒类与食用菌具有较高的相关性，其相关系数到达了 0.687464、茄类与其他品类相关性都不高，在单品之间的相互关系发现小米椒与红椒（1）存在较高相关性。然后使用 Apriori 关联算法深度研究品类之间的相互关系。进一步验证了皮尔逊系数的相关性信息。

针对问题二，需要探求销量与成本加成定价之间的相关性，考虑到该商超以成本加成定价方式来确定售价，分别对销量与成本加成定价、销量与加成率进行岭回归，发现销量与成本加成定价之间不存在明显的线性关系，销量与加成率进行岭回归之间存在明显线性关系，其中水生根茎类品类岭回归系数达到 6.18248218，考虑到需要对未来七天各品类进行自动定价与补货决策，我们计算各个品类的需求弹性系数，发现销量与售价也有较高的弹性关系。使用 ARIMA 算法对未来七天的销量、售价与成本进行预测，引入销量与售价弹性关系，以销售量为目标函数自定义优化算法来规划补货决策与自动定价。

针对问题三，与问题二有相似之处，通过计算各个单品的需求弹性系数发现其销量与售价并不存在明显的弹性关系，于是采用启发式算法进行规划补货决策与自动定价，采用 ARIMA 算法预测出 2023 年 7 月 1 日的销量、售价与成本作为模型初始化数据，导入遗传算法，以最佳总售价为适应度，品类价格变化与组合选取为变量，探求收益最大的补货决策与定价。得到预测的最优收益和为 754.440 元。

针对问题四，我们提出了需要针对每一段时间的具体损耗率、每一天的具体的批发数量、各单品的具体保鲜时长以及商超的实际储存量等问题中较为模糊的信息数据收集来寻求数据间的线性与非线性关系，并且提出季节性需求、销售趋势、天气状况、节假日、市场竞争和供应链可靠性这部分数据收集的重要性，使用层次分析法（AHP）对其重要性进行分析，发现季节性需求被认为是最重要的因素，其次是销售趋势、天气状况、节假日、市场竞争和供应链可靠性。

关键词： 皮尔逊系数　Apriori 关联算法　ARIMA 算法　遗传算法　层次分析法

一、 问题重述

1.1 问题背景

在生鲜商超中，一般蔬菜类商品的保鲜期都比较短，且品相随销售时间的增加而变差，大部分品种如当日未售出，隔日就无法再售。因此，商超通常会根据各商品的历史销售和需求情况每天进行补货。

由于商超销售的蔬菜品种众多、产地不尽相同，而蔬菜的进货交易时间通常在凌晨 3:00-4:00，为此商家须在不确切知道具体单品和进货价格的情况下，做出当日各蔬菜品类的补货决策。蔬菜的定价一般采用“成本加成定价”方法，商超对运损和品相变差的商品通常进行打折销售。可靠的市场需求分析，对补货决策和定价决策尤为重要。从需求侧来看，蔬菜类商品的销售量与时间往往存在一定的关联关系；从供给侧来看，蔬菜的供应品种在 4 月至 10 月较为丰富，商超销售空间的限制使得合理的销售组合变得极为重要。

1.2 问题提出

现在给出四个附件，附件 1 给出了某商超经销的 6 个蔬菜品类的商品信息；附件 2 和附件 3 分别给出了该商超 2020 年 7 月 1 日至 2023 年 6 月 30 日各商品的销售流水明细与批发价格的相关数据；附件 4 给出了各商品近期的损耗率数据。请根据附件和实际情况建立数学模型解决以下问题：

问题一：蔬菜类商品不同品类或不同单品之间可能存在一定的关联关系，请分析蔬菜各品类及单品销售量的分布规律及相互关系。

问题二：考虑商超以品类为单位做补货计划，请分析各蔬菜品类的销售总量与成本加成定价的关系，并给出各蔬菜品类未来一周(2023 年 7 月 1-7 日)的日补货总量和定价策略，使得商超收益最大。

问题三：因蔬菜类商品的销售空间有限，商超希望进一步制定单品的补货计划，要求可售单品总数控制在 27-33 个，且各单品订购量满足最小陈列量 2.5 千克的要求。根据 2023 年 6 月 24-30 日的可售品种，给出 7 月 1 日的单品补货量和定价策略，在尽量满足市场对各品类蔬菜商品需求的前提下，使得商超收益最大。

问题四：为了更好地制定蔬菜商品的补货和定价决策，商超还需要采集哪些相关数据，这些数据对解决上述问题有何帮助，请给出你们的意见和理由。

二、 问题分析

问题一：

针对问题一，我们探究品类单品及单品间的分布规律和相互关系。对该问题分析前，需要对数据进行预处理，对附件二退货部分销售类型，将退货数量的销售量取相反数并剔除打折部分，以确保数据的准确性。统计不同种类与单品在季节、月份等不同时间尺度下的销量。对蔬菜各品类的总销量、各品类不同时间尺度下的销量，绘制饼图分析各品类销售比例分布、绘箱

线图分析每天和每月的销售量分布、绘制折线图分析销售量随时间的变化分布、绘制柱状图分析不同季节下各品类的销售量分布与各个单品的销售总量分布，以此探求品类单品及单品间的分布规律。对于各品类及单品的相互关系的探求，采用皮尔逊系数初步进行研究其相关性，将平均销售量将销售量二值化后，使用 Apriori 算法寻找频繁项集并生成关联规则以寻求相互关系。

问题二：

问题分为两个部分，即分析各蔬菜品类的销售总量与成本加成定价的关系和各蔬菜品类未来一周(2023 年 7 月 1-7 日）的日补货总量和定价策略。以品类为单位，销售价格和成本能够计算出来，则可以得到成本加成定价的加成率，从而通过岭回归进行销售总量和成本加成定价加成率的拟合。据此使用时间序列模型预测未来一周的总量，依然保持加成比例不变，能够使商超收益最大。在此过程中需要考虑需求弹性模型，对销售价格与补货量进行计算，在该过程中，应加入损耗率进行求解。

问题三：

为解决问题三，首先需确定在 2023 年 6 月 24 日至 30 日的销售数据中，哪些单品是可售的（销售量不为零），构成备选单品列表。对每个可售单品建立数学模型，考虑历史销售数据、成本（批发价格和损耗率）和最小陈列量要求。运用数学优化技术，如线性规划，以最大化商超收益为目标，同时满足总单品数量（27-33 个）和最小陈列量的限制。针对每选定单品，基于成本加成定价方法确定价格，以最大化销售利润，依赖成本、历史销售、市场需求等。最终得到每个单品的补货量和定价策略，确保总补货量在合理范围内，同时满足市场需求。商超需考虑销售空间限制和陈列要求，确保商品适当展示。模型结果可评估并调整以适应市场和实际情况，从而在满足市场需求前提下，最大化蔬菜商品收益，兼顾销售空间利用和陈列规定。

问题四：

为了更好地制定蔬菜商品的补货和定价决策，商超需要采集每一段时间的具体损耗率、每一天的具体的批发数量、各单品的具体保鲜时长以及商超的实际储存量等问题中较为模糊的信息数据。这些数据的分析可以帮助商超制定准确的补货策略、调整定价以提高销售额和利润，并保持竞争力针对每一段时间的具体损耗率、每一天的具体的批发数量、各单品的具体保鲜时长以及商超的实际储存量等问题中较为模糊的信息数据收集来寻求数据间的线性与非线性关系，并且提出季节性需求、销售趋势、天气状况、节假日、市场竞争和供应链可靠性这部分数据收集的重要性，使用层次分析法（AHP）对其重要性进行分析，发现季节性需求被认为是最重要的因素，其次是销售趋势、天气状况、节假日、市场竞争和供应链可靠性。

三、 模型假设与符号说明

3.1 模型基本假设

(1) 假设各品类中各单品之间的相互关系不影响品类与品类之间的相互关系；
(2) 假设在预测的 7 天内销量平滑，不发生异常；
(3) 假设近期损耗率、补货量、销量之间存在线性关系；
(4) 假设设定的各准则的比较矩阵为适用于本数据；

3.2 符号说明

表1 符号说明

符号	含义	单位
$\bar{x}$	平均销量	kg
x_i	目标品类第 i 日单件销量	kg
n	单品数量	件
y	销量/平均销量	kg
x_j	成本定价加成率	%
β_j	岭回归曲线系数	
R	补货量	kg
L	平均损耗率	%
W_i	目标品类第 i 日平均批发价格预测	元/件
W	平均批发价格	元/件
P	收益	元
C	成本	元

四、 数据预处理

4.1 数据清洗

统计各个品类的销量在日期维度下时间序列的变化，发现在某些部分数据分布异常，花叶类数据尤为突出，其销量最高一条达 1200 千克一天，预计花叶类的这些离群值可能由于节日或者某种潮流导致。我们绘制箱线图进一步观察其箱线图，发现离群值散布在各个品类之间，结合两图可知，既部分特殊时间点出现销量异常的情况。在异常点过后恢复正常。将这类特殊时间点剔除不纳入后续的研究与分析。

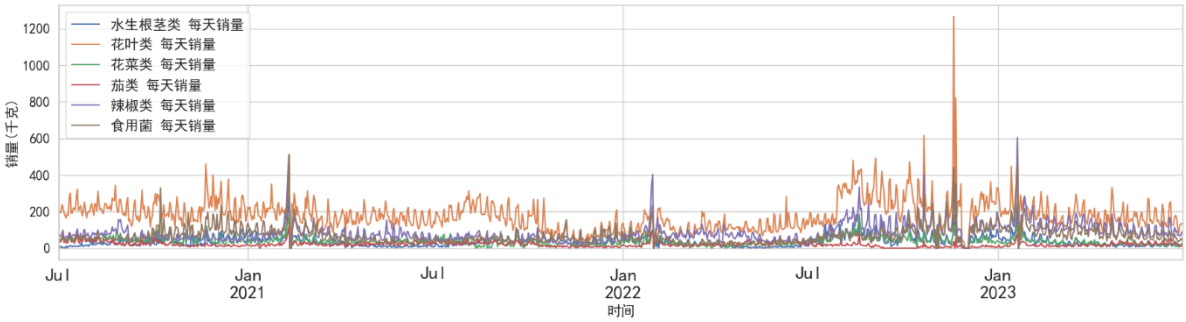

图 1 各品类在日期尺度下的销量折线图

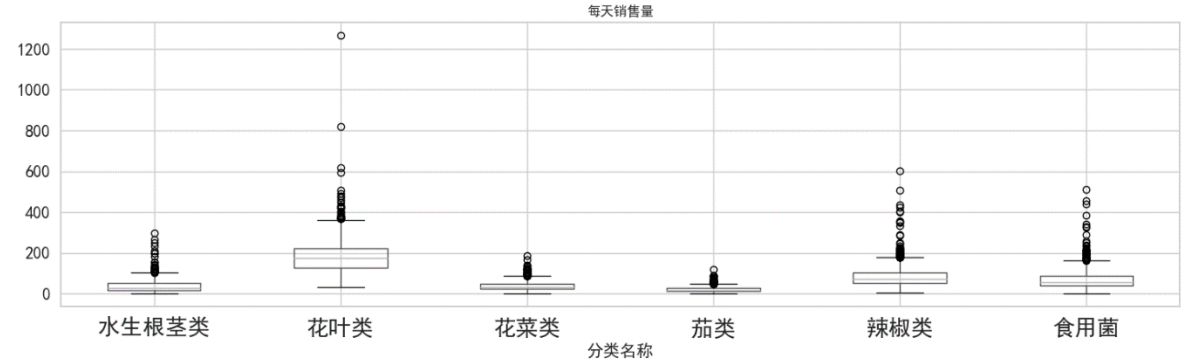

图 2 各品类在日期尺度下的销量箱线图

五、 模型建立与求解

5.1 问题一模型建立与求解

5.1.1 问题一求解思路

针对问题一需要进行数据预处理、销售量分析和关联性挖掘。首先，处理退货和打折销售数据以确保准确性。然后，统计销售量在不同时间尺度的分布，绘制图表分析品类销售比例、销售量分布和趋势。最后，利用皮尔逊相关系数和关联规则挖掘相互关系[1] 。这将帮助商超了解销售规律和购买模式，优化库存和销售策略，提高经济效益。

5.1.2 问题一模型建立

通过阅读题目我们首先对数据进行了预处理，将数据按照分类名称进行分组，并计算每个分类的总销量。

销售日期	扫码销售时间	销量(千克)	销售单价(元/千克)	销售类型	是否打折销售	单品名称
2020/7/1	15:07.9	0.396	7.6	销售	否	泡泡椒(精品)
2020/7/1	17:27.3	0.849	3.2	销售	否	大白菜
2020/7/1	17:33.9	0.409	7.6	销售	否	泡泡椒(精品)
2020/7/1	19:45.4	0.421	10	销售	否	上海青
…	…	…	…	…	…	…
2023/6/30	35:13.3	0.284	24	销售	否	西峡花菇(1)

表 1　　单品销售表

按销售量建立了饼图模型，箱线图模型，折线图；按不同蔬菜是否在打折下的销量分布、不同季节建立了柱状图；按总销量进行了柱状排列；使用皮尔逊相关系数计算不同蔬菜品类之间的销售量相关性，建立相关性矩阵。接着我们对数据进行二值化，根据平均值将销量分为 0 和 1，使用 Apriori 算法找到频繁项集[2] ，建立关联规则网络图。

5.1.3 问题一模型求解与分析

附件 2 收集的销售数据记录时间从 2020-7-1 到 2023-6-30，一共 1094 天。其中包括花叶类，花菜类，水生根茎类，茄类，辣椒类及食用菌六种品类及一共 1004 种单品。以花叶类中的单品为例，往往大品类内的单品会相互补充，例如单品编码为 102900005115793 的小白菜一旦库存不足，消费者可能会选择购买统一花叶类单品编码为 102900005115960 的大白菜作为补充，而非辣椒类或是茄类的其他蔬菜。即在同一品类内的单品呈现整体的负线性相关性。

接下来考察各品类之间销售量的关系，将不同品类的所有单品销售量求和，能够计算出各品类的总销售比例，其中花叶类占大部分销售，而茄类、花菜类和水生根茎类较少。图中表示花叶类的分布占大多数，其次是辣椒类、食用菌。

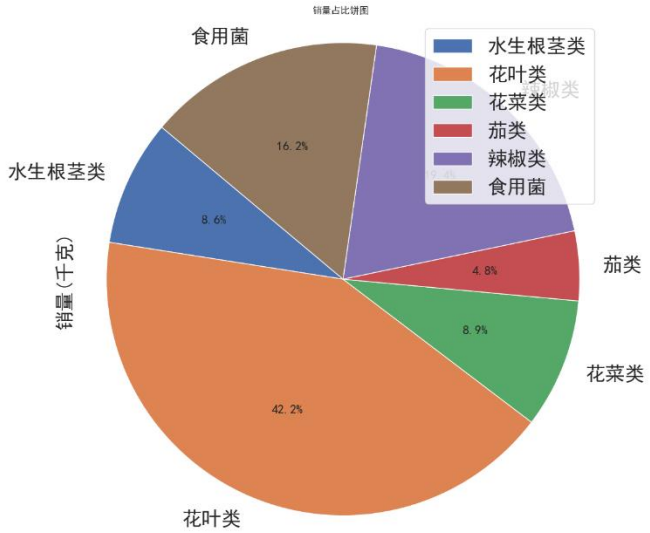

图 3 销量占比饼图

根据分类名称计算销售量总和，并在一个包含三个子图的图表中展示不同蔬菜品类在是否打折下的销售量分布。首先，代码根据分类名称分组并计算每个分类的销售总量。然后，使用柱状图在第一个子图中展示不同蔬菜品类的总销售量。接下来，在第二个和第三个子图中，使用筛选条件分别展示了打折销售和非打折销售的蔬菜品类销售量。最后，代码进行布局调整、保存图表和显示图表的操作。目的是提供对不同蔬菜品类在不同是否打折情况下销售量分布的可视化比较。由图可知打折与否对销量的影响不大。

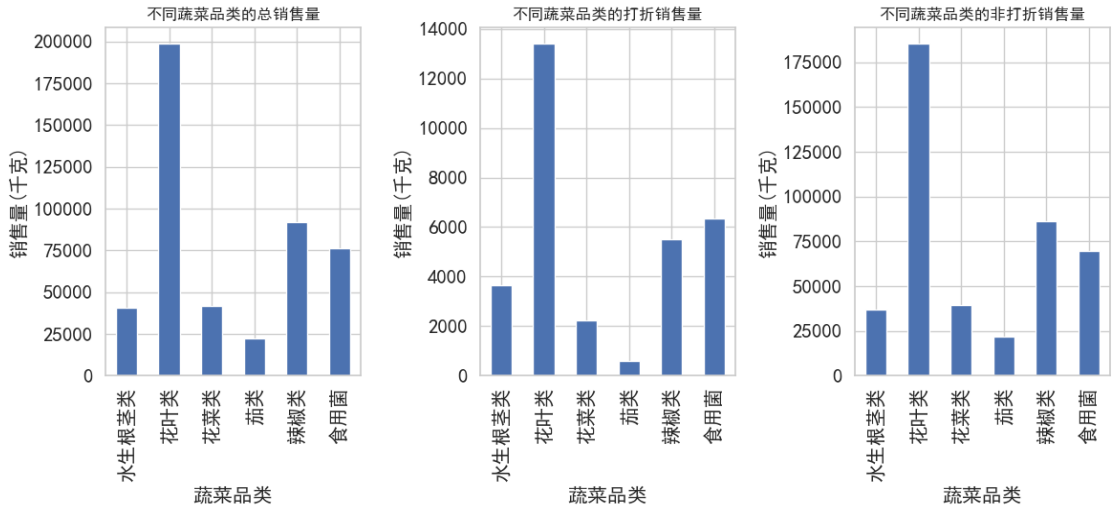

图 4 销量柱形图

将销售日期转换为季节，并根据季节和品类对销量进行汇总。然后通过柱状图展示不同季节下各品类的销量情况。首先将销售日期转换为日期时间对象，并提取出季节信息。接着按季节和品类对销量进行分组计算，并获取唯一的季节和品类值。然后，通过循环遍历每个季节，将各品类的销量绘制成柱状图，每个季节的柱状图使用不同的颜色。标签包括品类名称，并进行适当的旋转，以防止标签重叠。最后，图表被保存为文件，并通过调整布局确保图表的可读性。这段代码通过可视化展示销售量在不同季节和品类之间的变化，帮助我们了解销售的季节性模式和品类之间的差异。由图得知季节对花叶类、花菜类、茄类和辣椒类影响不大，对水生根茎类和食用菌类影响较大。

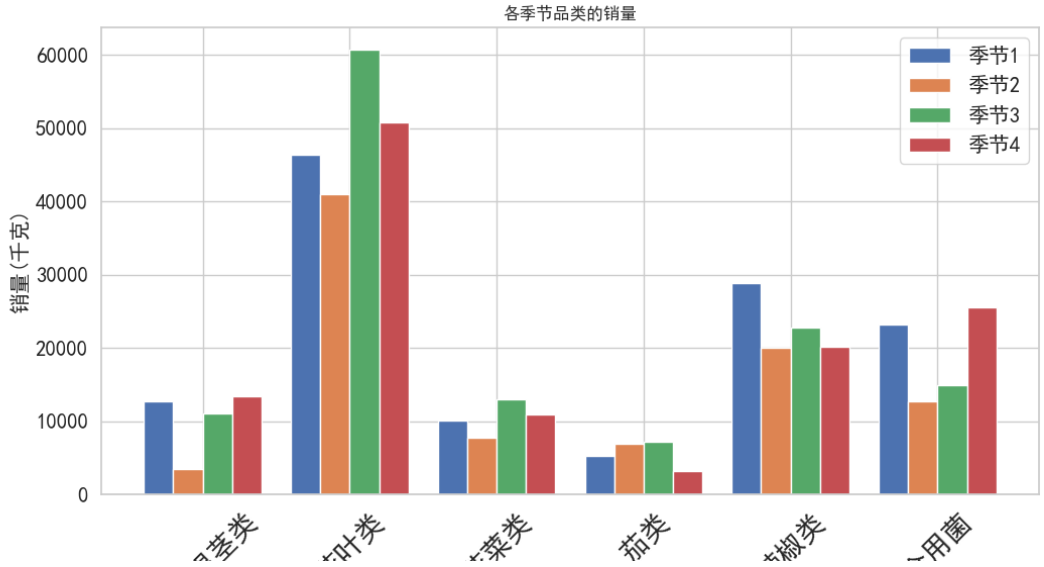

图 5 各季节品类销量柱状图

分析销售数据中单品的销售量，并选择销售量最高的前 50 个单品进行展示。它通过按单品名称分组，并计算每个单品的总销售量。然后，选择销售量最高的前 50 个单

品，并使用条形图展示它们的销售情况。图表可以用于比较和分析销售排名前 50 的蔬菜单品。其中芜湖青椒与西兰花的销售量较高。

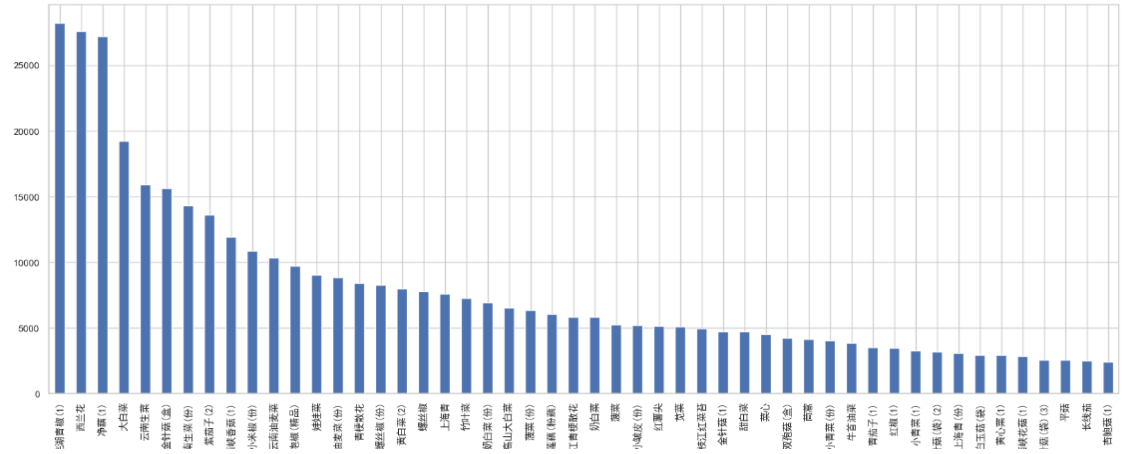

图 6 销量前 top50 单品柱状图

通过皮尔逊系数计算不同蔬菜品类之间的销售量相关性，并通过热力图进行可视化。目的是分析不同蔬菜品类之间的销售量相关性，并使用热力图将相关性矩阵可视化，从而更直观地了解不同品类之间的关系。

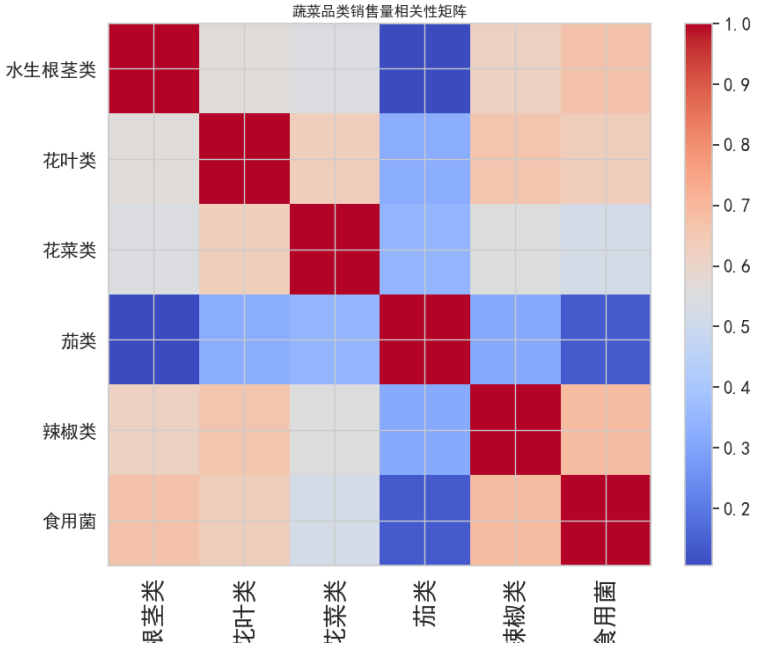

图 7 蔬菜品类销售量相关性矩阵

由图可以得，种类之间如茄类与花叶类相关性较弱，人们在购买二者中一种菜品就不会需要另一件菜品；辣椒类与花叶类相关性较强，二者同时被需要。如果不同蔬菜品类之间的销售量都呈现正相关关系，这意味着它们的销售量增长趋势大致相间的相似。根据这一观察，可以大概推测以下一些情况:

1.季节性趋势:正相关可能意味着这些蔬菜品类都受到季节性因素的影响。例如，如果其中一种蔬菜在某个季节销售量上升，其他品类的销售量也可能会相应上升，因为消费者在该季节更倾向于购买蔬菜。

2 市场趋势:正相关还可能反映了市场趋势或整体经济因素的影响。如果市场总体趋于增长，各个蔬菜品类的销售也可能随之增长。

3.销售策略和促销活动:正相关还可能与销售策略和促销活动有关。如果商超采取了一种特定的促销策略或销售战略,这可能导致不同蔬菜品类的销售都增加。

需要注意的是，虽然不同蔬菜品类之间的销售量正相关，但这并不代表它们的销售量增长幅度一定相同。某些品类可能会在特定情况下表现更好，而某些品类的销售增长可能较为温和。进一步的分析和数据收集可能需要进行，以更深入地了解这些趋势的原因和影响因素。

进行数据预处理

Setp1：通过‘分类名称’和‘销售日期’列对数据进行分组，并计算‘销量(千克)’列的总和。

Setp2：提取所有唯一的分类名称，并生成从最早销售日期到最晚销售日期之间的日期序列。

Setp3：统计出各个分类的日销售量。

Setp4： 通过各个品类的平均销量进行二值化，将销售量分为高销售量和低销售量。

关联性分析

Setp5：应用 Apriori 算法在转换后的数据中找到频繁项集，设置最小支持度为 0.01。

该方法旨在分析销售数据中不同类别之间的关联规则。它通过填充缺失值、二值化销售数据和使用 Apriori 算法计算频繁项集和关联规则，帮助了解销售项目之间的关联性。

二值化指标公式（平均值）：

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \qquad \textbf{(1)}$$

Apriori 算法: 使用 Apriori 算法对销售数据进行频繁项集挖掘和关联规则分析。首先，计算每个分类的销量平均值，并将销量进行二值化。然后，通过透视表将数据进行整理，以便进行

关联性分析。接着，使用 Apriori 算法找到频繁项集，即在整个数据集中频繁出现的项集。最后，使用关联规则函数找到满足提升度阈值的关联规则。这些分析可以帮助了解不同分类之间的销售关联性和相关规律。

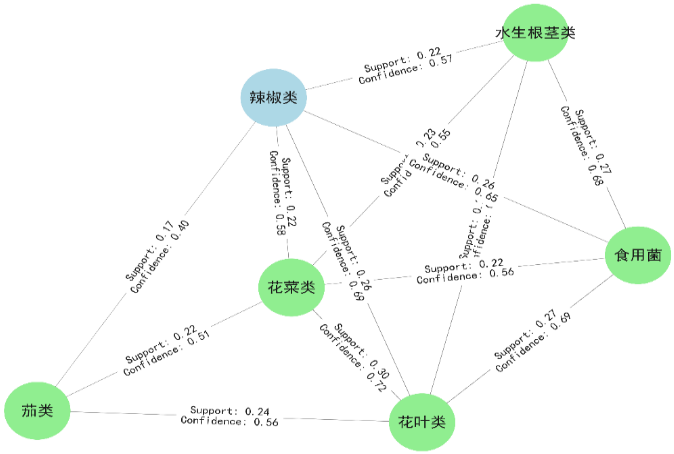

图 8 关联规则网络图

创建一个有向图，用于展示不同菜品关联规则之间的关系。从关联规则数据中提取出前 26 项规则，并将它们添加到网络图中。每个规则都用节点表示，节点的大小取决于其在规则中的关联度。关联规则之间的关系由边表示，边的宽度和颜色反映了其支持度和置信度，边上的标签显示了支持度和置信度的具体数值。节点的颜色和边的颜色反映了其在规则中的角色。最后，使用网络布局算法将节点和边放置在图中，并将图保存为图片进行展示。该图用途是可视化品类之间的关联规则网络结构。由图可以看出水生花类与食用菌的关联性较强，辣椒类与茄类关联性较弱。

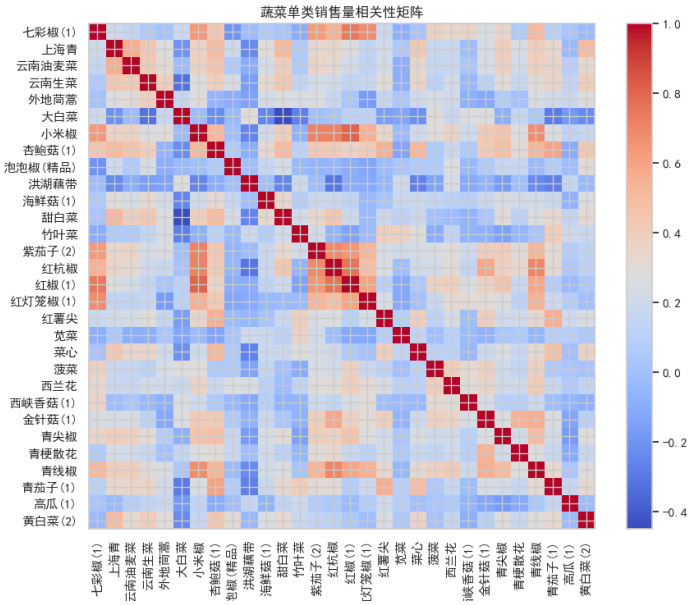

图 9 蔬菜单类销售量相关性矩阵

数值范围	相关程度
0-0.2	极小相关
0.2-0.4	较小相关
0.4-0.6	中等程度相关
0.6-0.8	较大相关
0.8-1.0	极大相关

表 2　相关程度表

相关系数用于测量两个变量之间的线性关系强度和方向。正相关系数表示变量正向变化，负相关系数表示反向变化。相关系数越接近 1 或-1，其相关性越强。

根据图中的数据，我们可以观察到一些有趣的结果。例如，甜白菜与大白菜之间的相关性为-0.4，表明它们之间存在负相关关系。这意味着当人们购买一种菜品时，就不太可能购买另一种菜品。或许这是因为这两种菜品在某种程度上具有相似的特征或用途，导致消费者更愿意选择其中的一种而不是两种都购买。另一个有趣的发现是红椒与小米辣之间的正相关性。这意味着当人们需要购买红椒时，他们也更可能购买小米辣。这可能是因为这两种调味品在烹饪中经常被同时使用，它们可能具有相似的风味或用途，消费者倾向于同时购买它们以满足烹饪需求。通过观察这些相关性，我们可以了解不同菜品之间的购买模式和关联程度，这对于制定商品搭配、销售策略以及库存管理都非常有帮助。这样的分析可以为企业提供有价值的信息，以更好地满足消费者需求并提高销售效益。

5.2 问题二模型建立与求解

5.2.1 问题二求解思路

对于问题二的第一部分，建立岭回归模型[6]，分析不同产品类别的平均批发价格与平均日销量之间的关系。读取相关数据并合并以获得每个类别每天的平均价格和销量数据。通过可视化方式绘制散点图和回归线，展示售价与销量之间的关系，并计算了岭回归的均方误差，以评估模型性能。对于问题二的第二部分，使用 ARIMA 模型预测了不同商品分类在 2023 年 7 月 1 日至 7 月 7 日的销售量[3]。读取销售数据，将销售日期转换为日期时间类型，并计算每个分类每天的销售总量。定义要预测的日期范围，初始化了一个数据集来存储预测结果。遍历每个分类，为其历史销售数据拟合 ARIMA 模型，根据拟合模型预测了未来日期范围内的销售量。这些分析和预测结果有助于商超优化定价策略和库存管理，以提高销售效益。

5.2.2 问题二模型建立

针对问题二第一问构建岭回归模型，分析不同产品类别的平均批发价格和平均日销量之间的关系。读入不同的数据，并合并数据以获得每个类别每天的平均价格和销量数据。接下来，通过绘制散点图和回归线的方式可视化不同类别的售价和销量关系，并计算了岭回归的均方误差。最后，保存了绘制的图像并打印了每个类别的岭回归系数。建立岭回归模型分析了平均批发价格对不同产品类别的平均日销量的影响，并通过可视化展示了它们之间的关系。分析商品销量与成本加成定价的加成率之间的关系。首先，从数据文件中读取销售数据、批发价格数

据、商品信息以及平均损耗率数据。然后，通过对数据进行合并和计算，得到销售总量、平均损耗率和成本等数据。接着，针对每个商品分类，利用岭回归模型拟合销量与成本加成定价的加成率之间的关系，并绘制散点图和拟合线。最终，可以通过观察图像来分析商品的成本加成定价与销量之间的关系。针对问题二第二问，由于要预测蔬菜品类未来一周(2023 年 7 月 1-7 日)的日补货总量和定价策略，该时间段并非法定节假日，突发情况发生率较小，所以可以使用 ARIMA（自回归整合移动平均）模型来预测不同商品分类的销售量。读取商品销售数据，将商品信息和销售数据整合在一起。将销售日期转换为日期时间类型，并计算每个分类在每天的销售总量。定义要预测的日期范围，然后初始化一个空的数据集来存储预测结果。遍历每个分类，为每个分类的历史销售数据拟合了 ARIMA 模型，根据拟合模型预测了未来日期范围内的销售量，并将结果添加到总的预测结果数据集中。将预测结果与原始商品分类信息合并并去除重复数据，以获得 2023 年 7 月 1 日至 7 月 7 日的销售量预测数据，以供进一步分析和可视化。这个过程允许对不同商品分类的销售趋势进行预测和分析。

5.2.3 问题二模型求解与分析

岭南回归模型：

岭回归（Ridge Regression）是回归方法的一种，来控制模型的复杂性，以防止过拟合，属于统计方法。在机器学习中也称作权重衰减。也有人称之为 Tikhonov 正则化。岭回归主要解决的问题是两种：一是当预测变量的数量超过观测变量的数量的时候（预测变量相当于特征，观测变量相当于标签），二是数据集之间具有多重共线性，即预测变量之间具有相关性。

$$y = \sum_{j=1}^{t} \beta_j x_j + \beta_0 \quad \textbf{(2)}$$

将销售数据和定价数据合并为一个数据集。根据"销售日期"和"单品编码"两个列将两个数据集合并在一起。然后，再次将合并后的数据集与第一个数据集根据相应的列进行合并，得到每日每个品类的平均定价和平均销量。为了绘制图形，先获取唯一的分类编码，调整子图之间的间距，再通过循环遍历不同的品类数据，在每个子图上绘制散点图和岭回归线。在每个子图中，首先绘制该品类的散点图，横轴为平均定价，纵轴为平均销量。然后，使 Ridge 回归模型进行岭回归分析。岭回归线用红色虚线表示。最后，计算预测值和实际值之间的均方误差（MSE）。在图形显示完毕后，输出每个品类的岭回归系数。通过绘制散点图和岭回归线来分析不同品类的售价与日均销量之间的关系，并计算了岭回归的系数和均方误差。最终结果以图像和输出系数的形式展示出来。

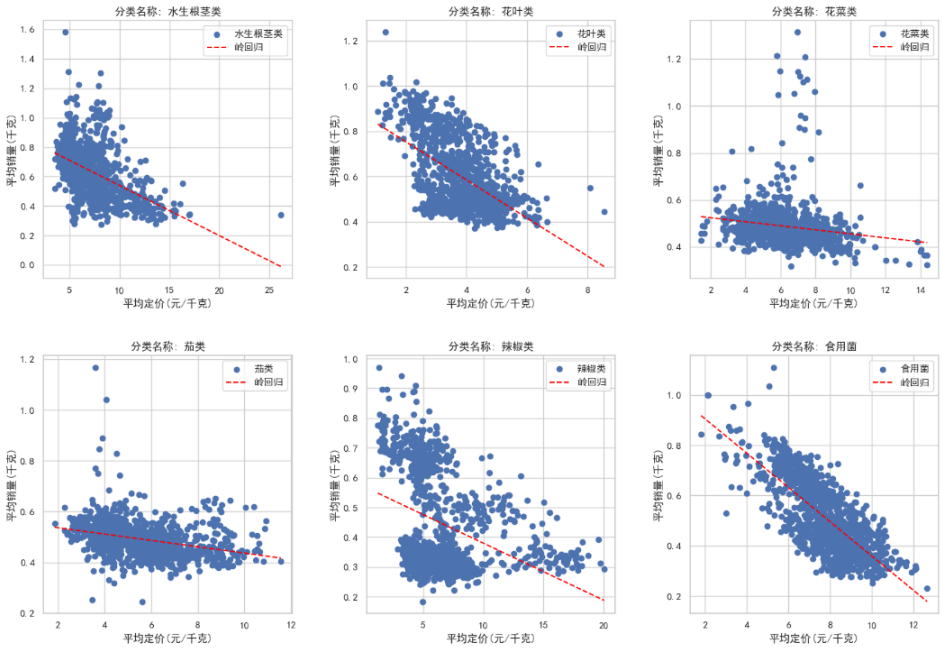

图 10 各品类的平均售价与日平均销量关系

分类名称	岭回归系数
水生根茎类	-0.03417075
花叶类	-0.08418141
花菜类	-0.00857735
茄类	-0.01247643
辣椒类	-0.01914358
食用菌	-0.06812304

表 3 品类岭回归系数（1）

由图表内容得出，相比六类中花菜类的岭回归系数绝对值较小，花菜类的平均销量和平均定价没有明显关系。而花叶类岭回归系数绝对值较大，花叶类的平均销量和平均定价成反比关系。

对成本加成定价的计算公式进行分析，处理四个数据文件，将销售数据、批发价格数据和商品信息合并，以及合并平均损耗率数据。接着，通过对数据进行分组统计，计算销售总量、平均损耗率和成本。

成本加成定价的计算公式：

$$售价 = 成本 + (成本 \times 加成率)$$

成本没法改动，故分析加成率

在得到销售总量、平均损耗率和成本数据后，通过考虑平均损耗率，计算成本加成定价的加成率。然后，针对每个不同的分类名称，使用岭回归模型拟合成本加成定价的加成率和销量之间的关系，并绘制散点图和拟合线。最后，通过子图布局的设置，将每个分类名称的散点图和拟合线显示在不同的子图。该图分析了不同分类名称下成本加成定价的加成率和销量之间的关系，并可视化展示了各品类的散点图和岭回归拟合线。

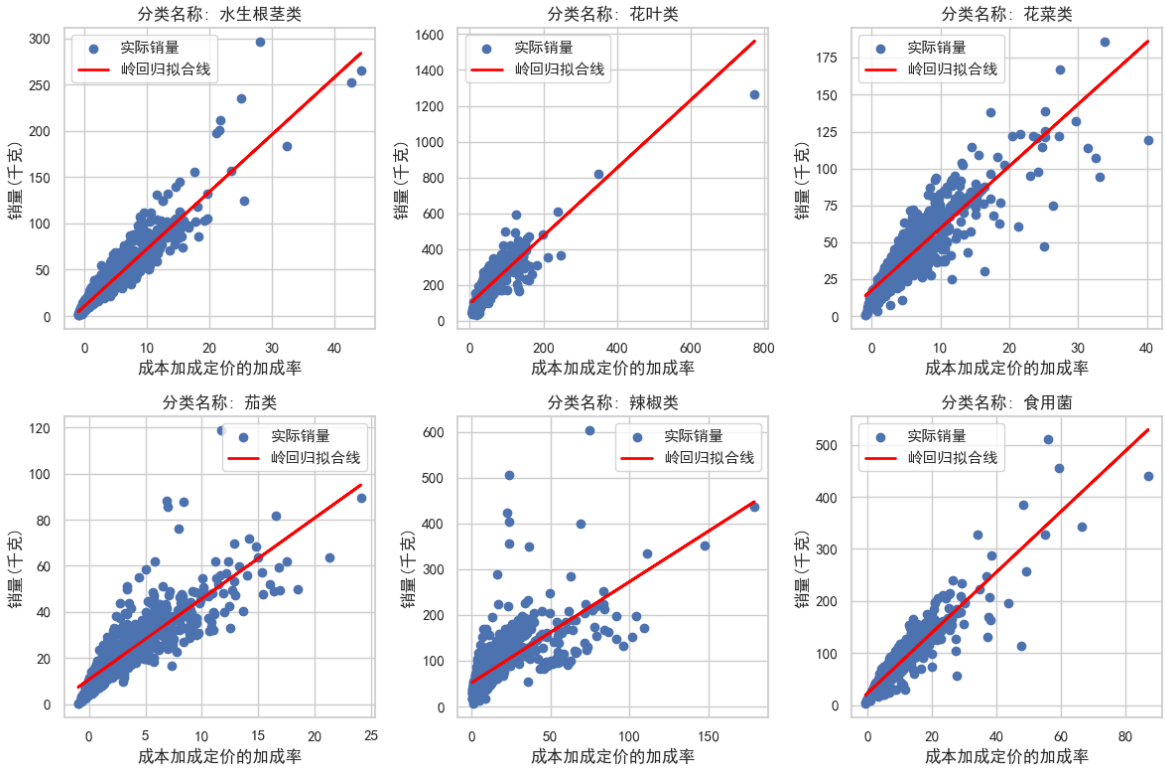

图 11 各品类的成本加成率和销量关系

分类名称	岭回归系数
水生根茎类	6.18248218
花叶类	1.89443031
花菜类	4.19965997
茄类	3.50344322
辣椒类	2.21456039
食用菌	5.80601339

表 4 品类岭回归系数（3）

根据图标内容，我们选择分析成本加成定价的加成率，这是通过计算公式来确定产品售价。通过岭回归系数，可以得知销量与成本加成定价的加成率之间存在正比关系。这意味着销售总量与成本加成定价之间存在一种直接关系，当成本加成定价增加时，销售总量也会随之增

加。这结果丰富了我们对价格影响销售的认识，暗示了在一定程度上价格可以通过成本加成定价的调整来影响销售总量。

销售日期	销售量（千克）	品类名称
2023/7/1	140.286032	花叶类
2023/7/2	143.028665	花叶类
…	…	…
2023/7/6	53.223103	食用菌
2023/7/7	53.251995	食用菌

表 5　2023 年 7 月 1 日到 7 日销售量预测

销售日期	平均批发价格（元/千克）	品类名称
2023/7/1	3.657434	花叶类
2023/7/2	3.684276	花叶类
…	…	…
2023/7/6	4.708648	食用菌
2023/7/7	4.708649	食用菌

表 6　2023 年 7 月 1 日到 7 日平均批发价格预测

销售日期	销售单价（元/千克）	品类名称
2023/7/1	4.988063	花叶类
2023/7/2	4.992779	花叶类
…	…	…
2023/7/6	11.418290	食用菌
2023/7/7	11.418259	食用菌

表 7　2023 年 7 月 1 日到 7 日销售单价预测

对 2023 年 7 月 1 日到 7 日的销售量与销售单价进行预测，发现当销售单价下降百分之几时，销售量会对有提升，反之当销售单价上涨百分之几时，销售量会下降。销售单价与销售量

之间具有反比例关系，由此可以引入弹性需求模型来预测蔬菜品类未来一周(2023 年 7 月 1-7 日)的日补货总量和定价策略。

$$C = \frac{R}{1-L\%} * W \quad (3)$$

$$X = \frac{R}{1-L\%} \quad (4)$$

$$P = C * X_j \quad (5)$$

弹性需求模型：

弹性需求模型，这是一种数学模型，用于描述价格变化对销量的影响。这种模型通常基于微观经济学中的概念，例如价格弹性性，它测量了价格变化对需求量变化的敏感程度。在数学建模中，使用弹性需求模型可以更好地理解和预测市场行为，从而制定更有效的策略。

销售日期	销售量（千克）	品类名称	平均批发价格（元/千克）	销售单价（元/千克）	平均损耗率（%）
2023/7/1	140.286032	花叶类	3.657434	4.988063	12.83
2023/7/2	143.028665	花叶类	3.684276	4.992779	12.83
…	…	…	…	…	…
2023/7/6	53.223103	食用菌	4.708648	11.41829	9.45
2023/7/7	53.251995	食用菌	4.708649	11.418259	9.45

表 8　2023 年 7 月 1 日到 7 日销售预测

品类名称	价格弹性系数
水生根茎类	-2.958727
花叶类	-11.455174
花菜类	-2.891346

茄类	-0.987076
辣椒类	-2.01078
食用菌	-3.287997

表 9　品类价格弹性系数

由表 8 可以得，花叶类（-11.455174）：这个类别的需求对价格变化非常敏感，属于弹性需求。当价格上升时，需求量可能大幅下降，而价格下降时，需求量可能大幅增加。食用菌（-3.287997）：食用菌类别的需求也比较敏感，属于弹性需求。价格上升可能导致需求下降，价格下降可能导致需求增加，但变化幅度较大。水生根茎类（-2.958727）、花菜类（-2.891346）、辣椒类（-2.010780）：这些类别的需求对价格变化也比较敏感，但相对于花叶类和食用菌来说，弹性较低。茄类（-0.987076）：茄类的需求对价格变化不太敏感，属于不太弹性需求。价格上升或下降时，需求量的变化幅度较小。

销售日期	销售量（千克）	品类名称	补货量	售价
2023/7/1	140.286032	花叶类	144.840459	5.486869
2023/7/2	143.028665	花叶类	147.672134	5.492057
…	…	…	…	…
2023/7/6	53.223103	食用菌	52.899826	12.560119
2023/7/7	53.251995	食用菌	52.928543	12.560085

表 10　预测未来一周内日补货量和定价表

5.3 问题三模型建立与求解

5.3.1 问题三求解思路

针对问题三，需要确定在 2023 年 6 月 24 日至 30 日的销售数据中，哪些单品是可售的，即它们的销售量在这段时间内不为零。这将构成可供选择的单品列表。针对每个可售单品，建立一个数学模型，考虑以下因素：单品的历史销售数据，包括销售量和销售价格。单品的成本信息，包括批发价格和损耗率。单品的最小陈列量要求。使用数学优化技术，例如线性规划或整数规划，以最大化商超的收益为目标函数，同时满足以下约束条件：可售单品总数在 27 到 33 之间。每个单品的订购量必须满足最小陈列量的要求。对于每个选定的单品，根据成本加

成定价方法，确定其价格，以最大程度地增加销售利润。定价策略可以基于成本、历史销售数据、市场需求等因素进行决策。根据优化模型的结果，得到每个单品的补货量和定价策略。确保总的补货数量在合理范围内，同时最大程度地满足市场需求。商超还需考虑蔬菜商品的销售空间限制和陈列安排，以确保商品能够适当陈列和销售。商超可以评估模型的结果并进行必要的调整，以适应市场变化和实际情况。将使商超能够在满足市场需求的前提下，最大化其蔬菜类商品的收益，同时确保了销售空间的有效利用和陈列要求的满足。

5.3.2 问题三模型建立

针对问题三，首先需要确定在 2023 年 6 月 24 日至 30 日的销售数据中，哪些单品是可售的，即它们的销售量在这段时间内不为零。这将构成可供选择的单品列表。随后，我们采用启发式模型来处理单品的补货计划和定价策略。对于每个可售单品，我们建立了启发式模型，考虑了多个关键因素，包括单品的历史销售数据（包括销售量和销售价格）、成本信息（批发价格和损耗率）以及最小陈列量要求。然后使用启发式算法，如遗传算法等[4]，来优化补货量和定价策略，以最大化商超的收益。目标是找到每个单品的最佳补货量和价格，以在总补货数量在合理范围内的情况下，最大化销售利润。这个综合的启发式模型允许我们灵活地处理不同单品的特性和市场需求。除了补货计划和定价策略，商超还需考虑销售空间的限制和陈列要求，以确保商品能够适当陈列和销售。最后商超可以根据模型的结果进行评估，并根据实际市场变化和需求调整决策，以确保在满足市场需求的前提下，最大化蔬菜类商品的收益。这个启发式模型将帮助商超更有效地管理蔬菜类商品的补货和定价策略，以实现最佳经济效益。

5.3.3 问题三模型求解与分析

品类名称	销售日期	销量（千克）	销量总额	平均销售价格	单品名称
水生根茎类	2023/6/24	5.757	92.112	16	净藕(1)
水生根茎类	2023/6/24	1.955	31.28	16	高瓜(1)
…	…	…	…	…	…
食用菌	2023/6/30	3	9	3	海鲜菇(包)

表 11　销售日期筛选表

单品名称	价格弹性系数
高瓜(1)	-0.05944099
洪湖藕带	-0.1521275

…	…
双孢菇(盒)	1.10707
蟹味菇与白玉菇双拼(盒)	0.2397228

表 12　48 类单品的弹性系数

通过对每个单品的售价，平均销售价格的处理，将其拟合成线性回归模型，处理数据并提取出其价格弹性系数，发现其弹性系数都很低，价格变化对销售量的影响很小，因此不适合直接用于定价策略的决策，并不适合问题二的做法，所以引入了启发式算法进行补货量与定价策略的决策。使用 ARIMA（差分自回归移动平均模型）来初始化销售数据，特别是当天的销量和售价等信息， ARIMA 可以捕捉趋势和季节性。这种综合方法将统计分析、启发式算法和时间序列模型相结合，以更好地应对商超蔬菜销售问题，为商超提供可靠的决策支持，以最大化其收益。

遗传算法：

遗传算法是一种基于启发式方法的算法。启发式方法是一种问题解决方法，它不依赖于严格的数学规则或完全的搜索，而是借助经验、规则和直观的方法来找到可能的解决方案。启发模型通常用于解决复杂问题，特别是在问题的搜索空间非常大或难以精确建模时。

为了可售单品总数控制在 27-33 个，且各单品订购量满足最小陈列量 2.5 千克的要求。使用 ARIMA 算法预测出每个品类的销量、销售价格和批发价格，进行销售收益的计算，使用正确的收益计算方式，将单品数限制在 27-33 之间。运用遗传算法数据进行一定的变异，然后根据适应度函数评定每个销售组合对本次迭代的适应度，留取适应度最高的组合进行继续迭代获取出最佳组合，总收益为 754.460148674059 元。

销售组合总收益计算公式：

$$P = \sum_{i}^{n} (X * C * (1 + x_i)) - \left(\frac{X}{1-L\%}\right) * W_i \qquad (6)$$

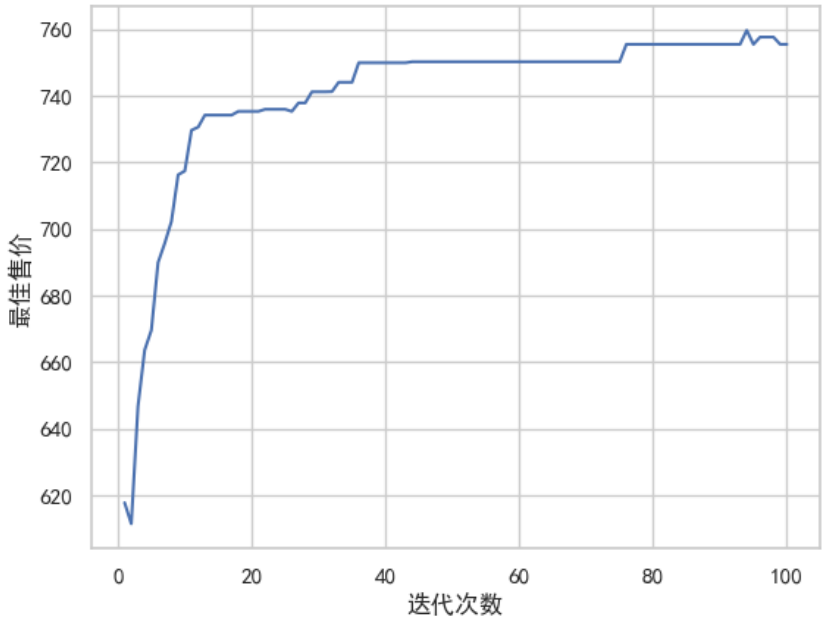

图 12　迭代次数与最佳售价关系图

单品名称	补货量	平均销售价格预测	收益
高瓜(1)	2.895194	16	6.209625
洪湖藕带	3.213994	25.891431	13.976127
…	…	…	…
双孢菇(盒)	9.797658	5.5	15.426339
蟹味菇与白玉菇双拼(盒)	2.760906	5.8	5.839041
总收益			754.460148674059

表 13　单品补货量、定价和收益

5.4 问题四模型建立与求解

5.4.1 问题四模型求解与分析

（1）针对每一段时间的具体损耗率：

我们在计算补货量时，对销量进行修正以确定补货量：

$$补货量 = （1 - 损耗率） \times 销量 \quad (7)$$

补货量直接决定了销售成本：

$$\text{成本} = \text{补货量} \times \text{批发价格}$$

补货量直接决定了销售成本，在问题二中，通过定义计算收益函数对各品类的补货总量和定价策略将有更清晰的判断

（2）每一天的具体的批发数量、各单品的具体保鲜时长以及商超的实际储存量：

附件 4 中给出了各单品每日的批发价格，但没有给出具体批发数量，题目中声明大部分商品无法隔天销售，却没有指明是哪些商品。在问题二、三的计算中，对商品的补货数量与销量存在一个很模糊的界限，大多以损耗率修正销量作为商品的补货数量。明确各单品每一天的具体的批发数量、各单品的具体保鲜时长以及商超的实际储存量，可以在时间上对商品的补货有更明确的标准，对蔬果产品的新鲜度等指标也可以的定义。在多个指标的情况下，我们可以更为清晰的分析各品类以及单品的商品售价与销量之间的非线性关系。

（3）当天天气状况以及节假日等信息：

天气状况以及节假日等因素可以直接影响到商超的人流量，销售趋势、季节性需求等可以对各个商品的销量起到决定性因素，市场竞争、供应链可靠性等更是从可以视为突发情况导致各品类销量的短期变化。

为了解决问题探究变量之间对蔬菜类商品的自动定价与补货决策的影响，我们使用了层次分析法（AHP）来确定以上各个不同因素的权重[5]，对各个待定底层准则的加权矩阵进行拟定，以便制定决策或评估各种方案的影响。

通过 AHP 分析，我们确定了在制定蔬菜商品的补货和定价决策中，不同因素的相对重要性。季节性需求被认为是最重要的因素，其次是销售趋势、天气状况、节假日、市场竞争和供应链可靠性。这些权重可以指导决策，明确我们需要对各商品的季节性销售信息的相关数据的优先需求，其次是销售趋势、天气状况、节假日、市场竞争和供应链可靠性相关数据。

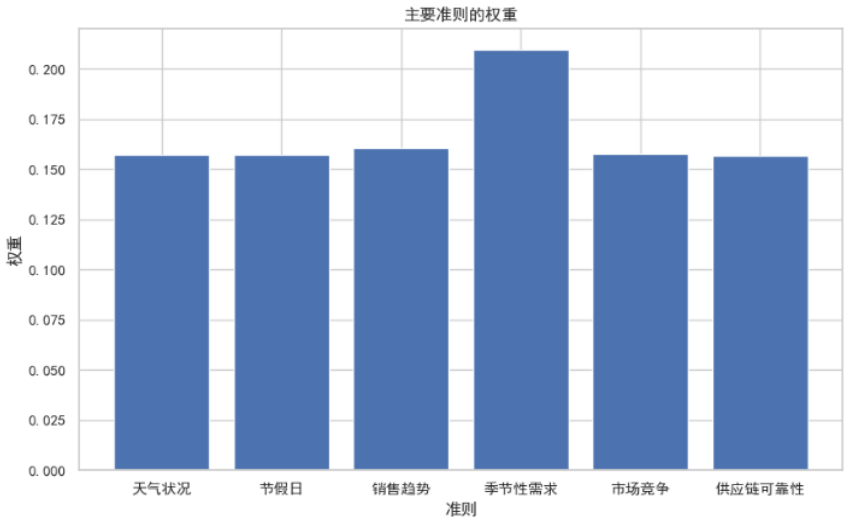

图 13　AHP 分析下的各个主要准则权重

六、 模型评价与推广

6.1 模型的优点

（1）问题一使用了饼图、箱型图、直方图、皮尔逊系数和 Apriori 关联算法等模型。这些模型的优点包括饼图直观展示占比、箱型图检测异常、直方图展示分布、皮尔逊系数量化相关性和 Apriori 关联算法挖掘关系；

（2）问题二中的自定义优化算法，引入了弹性需求模型考虑了实际情况定义真实的最大销量，根据具体问题和约束进行调整，模型有更好的可调整能力且有较高的可解释性；

（3）问题三中定义遗传算法结合 ARIMA，将 ARIMA 时序预测模型作为遗传算法的初始输入可以加速算法的收敛，降低计算成本，并提供有关时间序列数据结构的重要信息；

（4）问题四模型通过层次分析法（AHP）对不同因素的重要性进行分析，包括季节性需求、销售趋势、天气状况、节假日、市场竞争和供应链可靠性，综合考虑了多个因素对销售的影响。提供决策支可以帮助商超更好地了解销售环境和关键因素，为制定合理的补货策略和定价策略提供了重要的决策支持；

6.2 模型的不足

（1）问题一中如饼图无法提供详细信息、箱型图无法解释异常值原因、直方图可能无法捕捉细微变化、皮尔逊系数局限于线性关系和 Apriori 算法对数据特征和参数设置敏感；

（2）问题二中数据质量要求高，对数据销量与售价之间要求存在较高的弹性关系，未考虑市场竞争因素对销量和定价的影响，以及模型复杂性可能需要更多的资源和时间。因此，需要进一步改进和精化模型，考虑更多非线性因素、市场竞争，以提高模型的准确性和实用性；

（3）问题三中预测时效性受挑战、忽略市场竞争影响、复杂性较高；

（4）问题四中层次分析法的两两矩阵缺乏认证，不确保它的有效性，不能保证模型的泛用性；

6.3 模型的推广

问题二的综合模型可以广泛应用于多个领域，包括零售、电子商务、餐饮、制造、金融、市场营销、物流和医疗保健等。它帮助组织通过关联性算法、时序分析、需求弹性模型和自定义优化算法来自动化商品定价和库存管理，以适应市场需求，提高销售利润和效率。模型的灵活性和可定制性使其适用于不同行业和复杂商业情境，可优化价格策略和库存控制，提升组织的竞争力。

参考文献

[1] 王卓宇. 数据间稳定相关关系的研究[D]. 西安电子科技大学,2021.DOI:10.27389/d.cnki.gxadu.2020.000578.刘志梅.数学建模与高职数学教学的深度融合[J].佳木斯职业学院学报,2023,39(03):152-154.

[2] 张梦琦. 基于 Apriori 算法的关联规则分析[D]. 大连理工大学,2022.DOI:10.26991/d.cnki.gdllu.2021.001178.

[3] 冯剑,姚罕琦,黄啸虎等.ARIMA 算法在工业控制器故障预测的应用[J].自动化仪表,2022,43(11):62-67.DOI:10.16086/j.cnki.issn1000-0380.2021120034.

[4] 郭景阳. 差分交叉遗传算法及其在结构和惯容系统优化设计中的应用[D].烟台大学,2023.DOI:10.27437/d.cnki.gytdu.2023.000378.

[5] 唐若宓.基于层次分析法（AHP）的技工院校学生学习质量评价模型的构建——以食品加工与检验专业为例[J].中国食品,2023(04):69-71.骆成蹊.使用层次分析法（AHP）解决异地灾备数据中心选址问题[J].现代电视技术,2022(09):110-114.

[6] 郭恒亮,李晓,付羽等.基于核岭回归算法的 PROSAIL 模型反演高空间分辨率叶面积指数[J].草业学报,2022,31(12):41-51.

附录

附录 1

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
import networkx as nx
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
import warnings
warnings.filterwarnings("ignore")
plt.rcParams['axes.unicode_minus'] = False
sns.set(style="whitegrid")
plt.rcParams['font.sans-serif'] = ['SimHei']
[154]
df1 = pd.read_csv('data/附件 1.csv').iloc[:,1:]
df2 = pd.read_csv('data/附件 2.csv').iloc[:,1:]
df3 = pd.read_csv('data/附件 3.csv').iloc[:,1:]
df4 = pd.read_csv('data/附件 4.csv').iloc[:,1:]
```

第一题代码

```
[830]
q1_df = pd.merge(df2, df1, on='单品编码', how='left')
[4]
q1_df
[5]
q1_df.loc[q1_df['销售类型'] == '退货', '销量(千克)'] = -q1_df.loc[q1_df['销售类
型'] == '退货', '销量(千克)']
[6]
sales = q1_df.groupby(['分类名称'])['销量(千克)'].sum().reset_index()
sales.plot(kind='pie', y='销量(千克)', labels=sales['分类名称'], au-
topct='%1.1f%%', startangle=140, figsize=(12, 12))
plt.axis('equal')
plt.title('销量占比饼图')
plt.savefig('./Image/销量占比饼图.png')
plt.show()

[7]
# 将销售日期列转换为日期时间对象
try:
    q1_df['销售日期'] = pd.to_datetime(q1_df['销售日期'])
except:
    pass

# 按品类、日期进行分组并计算销售量的总和
```

```
daily_sales = q1_df.groupby(['分类名称', '销售日期'])['销量(千克)'].sum().re-
set_index()

# 设置日期作为索引
daily_sales.set_index('销售日期', inplace=True)

# 创建不同时间维度的箱线图
fig, axes = plt.subplots(2, 1, figsize=(15, 10))

# 每天销售量箱线图
daily_sales.boxplot(column='销量(千克)', by='分类名称', ax=axes[0])
axes[0].set_title('每天销售量')

# 每月销售量箱线图
monthly_sales = daily_sales.groupby(['分类名称', pd.Grouper(freq='M')])['销量
(千克)'].sum().reset_index()
monthly_sales.boxplot(column='销量(千克)', by='分类名称', ax=axes[1])
axes[1].set_title('每月销售量')

# 调整布局
plt.tight_layout()
plt.savefig('Image/销售量箱线图比较.png')
plt.show()
```

```
[8]
# 将销售日期列转换为日期时间对象
try:
    q1_df['销售日期'] = pd.to_datetime(q1_df['销售日期'])
except:
    pass

Time_changes = q1_df.copy()

# 设置销售日期列为索引
Time_changes.set_index('销售日期', inplace=True)

# 根据不同品类进行分组
grouped = Time_changes.groupby('分类名称')
[9]
# 定义函数来绘制销量曲线
def plot_sales_by_time(df, title, ax):
    # 月度销量
    monthly_sales = df.resample('M')['销量(千克)'].sum()

    # 每天销量
    daily_sales = df.resample('D')['销量(千克)'].sum()

    monthly_sales.plot(ax=ax[0], label=f'{title} 月度销量')
```

```
    daily_sales.plot(ax=ax[1], label=f'{title} 每天销量')

# 创建一个包含4个子图的图形
fig, axes = plt.subplots(2, 1, figsize=(18, 10))
plt.subplots_adjust(hspace=0.5)

# 遍历不同品类并绘制销量曲线在同一个子图中
for category, group in grouped:
    plot_sales_by_time(group, category, axes)

# 添加图例
for ax in axes:
    ax.legend()

plt.xlabel('时间')
plt.ylabel('销量(千克)')
plt.tight_layout()
plt.savefig('Image/销售量折线图.png')
plt.show()
```

```
[10]
# 根据分类名称计算销售量总和
category_sales = q1_df.groupby('分类名称')['销量(千克)'].sum()

# 创建一个包含三个子图的图表
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(12, 6))

# 第一个子图：所有品类的销售量
category_sales.plot(kind='bar', ax=ax1)
ax1.set_xlabel('蔬菜品类')
ax1.set_ylabel('销售量(千克)')
ax1.set_title('不同蔬菜品类的总销售量')

# 第二个子图：打折销售的销售量
discounted_category_sales = q1_df[q1_df['是否打折销售'] == '是'].groupby('分类名称')['销量(千克)'].sum()
discounted_category_sales.plot(kind='bar', ax=ax2)
ax2.set_xlabel('蔬菜品类')
ax2.set_ylabel('销售量(千克)')
ax2.set_title('不同蔬菜品类的打折销售量')

# 第三个子图：非打折销售的销售量
non_discounted_category_sales = q1_df[q1_df['是否打折销售'] == '否'].groupby('分类名称')['销量(千克)'].sum()
non_discounted_category_sales.plot(kind='bar', ax=ax3)
ax3.set_xlabel('蔬菜品类')
ax3.set_ylabel('销售量(千克)')
ax3.set_title('不同蔬菜品类的非打折销售量')
```

```
plt.tight_layout()  # 保证子图之间的间距合适
plt.savefig('Image/不同蔬菜品类在是否打折下的销售量分布.png')
plt.show()

[11]
# 将销售日期转换为季节
q1_df['销售日期'] = pd.to_datetime(q1_df['销售日期'])
q1_df['季节'] = q1_df['销售日期'].dt.quarter

# 按季节和品类对销量进行汇总
seasonal_sales = q1_df.groupby(['季节', '分类名称'])['销量(千克)'].sum().re-
set_index()

# 获取季节和品类的唯一值，以确定柱子的位置
seasons = seasonal_sales['季节'].unique()
categories = seasonal_sales['分类名称'].unique()
category_count = len(categories)
bar_width = 0.2  # 柱子的宽度

# 绘制各季节品类的销量
fig, ax = plt.subplots(figsize=(12, 6))

for i, season in enumerate(seasons):
    season_data = seasonal_sales[seasonal_sales['季节'] == season]
    x = np.arange(len(categories)) + i * bar_width
    ax.bar(x, season_data['销量(千克)'], width=bar_width, label=f'季节{sea-
son}')

ax.set_xlabel('品类')
ax.set_ylabel('销量(千克)')
ax.set_title('各季节品类的销量')
ax.legend()

# 设置 x 轴标签
ax.set_xticks(np.arange(len(categories)) + ((len(sea-
sons) - 1) / 2) * bar_width)
ax.set_xticklabels(categories, rotation=45)

plt.savefig('Image/不同季节下的销量.png')
plt.tight_layout()
plt.show()

[12]
# 根据单品名称计算销售量总和
product_sales = q1_df.groupby('单品名称')['销量(千克)'].sum()

# 选择销售量最高的前 N 个单品
```

```
top_n = 50
top_product_sales = product_sales.nlargest(top_n)

# 创建一个包含三个子图的图表
fig, (ax1) = plt.subplots(1, 1, figsize=(20, 8))

total_sales = product_sales[top_product_sales.index]
total_sales.plot(kind='bar', ax=ax1)
ax3.set_xlabel('蔬菜单品')
ax3.set_ylabel('总销售量(千克)')
ax3.set_title(f'销售量前{top_n}的蔬菜单品（总销售量）')

plt.savefig(f'Image/销售量前{top_n}的蔬菜单品（总销售量）.png')
plt.tight_layout()  # 保证子图之间的间距合适
plt.show()

correlation_matrix
[1086]
correlation_matrix
[831]
# 使用皮尔逊相关系数计算不同蔬菜品类之间的销售量相关性
correlation_matrix = q1_df.pivot_table(index='销售日期', columns='分类名称
', values='销量(千克)', aggfunc='sum').corr()

# 绘制热力图
plt.figure(figsize=(10, 8))
plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='nearest')
plt.colorbar()
plt.xticks(range(len(correlation_matrix)), correlation_matrix.columns, rota-
tion=90)
plt.yticks(range(len(correlation_matrix)), correlation_matrix.columns)
plt.title('蔬菜品类销售量相关性矩阵')
plt.savefig(f'Image/蔬菜品类销售量相关性矩阵.png')
plt.show()

[83]
Aprioridf = q1_df.copy()

Aprioridf = Aprioridf.groupby(['分类名称','销售日期'])['销量(千克)'].sum().re-
set_index()
# 将销售时间列转换为 datetime 类型
Aprioridf['销售日期'] = pd.to_datetime(Aprioridf['销售日期'])
all_categories = Aprioridf['分类名称'].unique()
all_minutes = pd.date_range(start=Aprioridf['销售日期'].min(), end=Aprioridf['
销售日期'].max(), freq='D')
index = pd.MultiIndex.from_product([all_categories, all_minutes], names=['分类
名称', '销售日期'])
filled_df = pd.DataFrame(index=index).reset_index()
# 将填充的数据与原始数据进行合并，此时销售时间列的数据类型相同
```

```
merged_df = filled_df.merge(Aprioridf, on=['分类名称', '销售日期'], how='left')
# 填充缺失的销量为0
merged_df['销量(千克)'].fillna(0, inplace=True)
Aprioridf = merged_df.copy()
[87]
Aprioridf
[89]
# 数据预处理：将销量二值化
average_sales_by_category = Aprioridf.groupby('分类名称')['销量(千
克)'].mean() # 计算每个分类的销量平均值
# 根据平均值将销量划分为0和1
Aprioridf['销量(千克)'] = Aprioridf.apply(lambda row: 0 if row['销量(千
克)'] < average_sales_by_category[row['分类名称']] else 1, axis=1)

# 使用pivot_table将数据进行透视，以便进行关联性分析
basket = (Aprioridf.pivot_table(index='销售日期', columns='分类名称', values='
销量(千克)', aggfunc='sum'))

# 使用Apriori算法找到频繁项集
frequent_itemsets = apriori(basket, min_support=0.01, use_colnames=True)

# 使用关联规则函数找到关联规则
rules = association_rules(frequent_itemsets, metric='lift', min_threshold=1.0)
[108]
rules
[105]
# 创建一个有向图，并设置图的大小
G = nx.DiGraph()

# 添加关联规则到网络图中
for _, row in rules[:26].iterrows():
    antecedents = list(row['antecedents'])
    consequents = list(row['consequents'])
    support = row['support']
    confidence = row['confidence']

    # 添加节点和边，同时传递关联度信息
    G.add_node(', '.join(antecedents), size=1000)
    G.add_node(', '.join(consequents), size=1000)
    G.add_edge(', '.join(antecedents), ', '.join(consequents), support=sup-
port, confidence=confidence)

# 设置节点大小和颜色
node_sizes = [G.nodes[node]['size'] for node in G.nodes]
node_colors = ['lightblue' if node in antecedents else 'light-
green' for node in G.nodes]

# 设置边的宽度和颜色，以及标签
edge_weights = [G.edges[edge]['support'] for edge in G.edges]
```

```
edge_colors = ['gray' if weight < 0.5 else 'red' for weight in edge_weights]
edge_labels = {(edge[0], edge[1]): f"Support: {G.edges[edge]['sup-
port']:.2f}\nConfidence: {G.edges[edge]['confi-
dence']:.2f}" for edge in G.edges}

# 绘制网络图
pos = nx.spring_layout(G, seed=42)
nx.draw_networkx_nodes(G, pos, node_size=node_sizes, node_color=node_colors)
nx.draw_networkx_edges(G, pos, width=edge_weights, edge_color=edge_colors, al-
pha=0.7)
nx.draw_networkx_labels(G, pos, font_size=8)


# 添加边的标签
nx.draw_networkx_edge_labels(G, pos, edge_labels, font_size=6)

plt.axis('off')
plt.title("关联规则网络图")
plt.savefig('./image/品类关联规则网络图.png',dpi=1000)
plt.show()

[114]
selected_vegetables = q1_df['单品名称'].unique()[:30]

# 使用 pivot_table 计算不同蔬菜单品之间的销售量相关性
correlation_matrix = q1_df[q1_df['单品名称'].isin(selected_vegeta-
bles)].pivot_table(index='销售日期', columns='单品名称', values='销量(千
克)', aggfunc='sum').corr()

# 绘制热力图
plt.figure(figsize=(10, 8))
plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='nearest')
plt.colorbar()
plt.xticks(range(len(correlation_matrix)), correlation_matrix.columns, rota-
tion=90)
plt.yticks(range(len(correlation_matrix)), correlation_matrix.columns)
plt.title('蔬菜单类销售量相关性矩阵')
plt.savefig(f'Image/蔬菜部分单品销售量相关性矩阵.png')
plt.show()
```

第二题代码

```
[260]
import pandas as pd
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.arima.model import ARIMA
```

成本加成定价的计算公式： 售价 = 成本 + (成本 × 加成率)，成本没法改动，故分析加成率

[323]
```
# 读取数据
df1 = pd.read_csv('data/附件 1.csv').iloc[:, 1:]
df2 = pd.read_csv('data/附件 2.csv').iloc[:, 1:]
df3 = pd.read_csv('data/附件 3.csv').iloc[:, 1:]
df4 = pd.read_csv('data/附件 4.csv').iloc[:, 1:]

# 统一列名
df1 = df1.rename(columns={'单品编码': '单品编码', '单品名称': '单品名称', '分类编码': '分类编码', '分类名称': '分类名称'})
df2 = df2.rename(columns={'销售日期': '日期', '单品编码': '单品编码', '销量(千克)': '销量(千克)', '销售单价(元/千克)': '销售单价(元/千克)', '销售类型': '销售类型', '是否打折销售': '是否打折销售'})
df3 = df3.rename(columns={'日期': '日期', '单品编码': '单品编码', '批发价格(元/千克)': '批发价格(元/千克)'})
df4 = df4.rename(columns={'小分类编码': '分类编码', '小分类名称': '分类名称', '平均损耗率(%)_小分类编码_不同值': '平均损耗率(%)'})

# 合并销售数据与批发价格数据
df_sales = df2.merge(df3, on=['日期', '单品编码'], how='inner')

# 合并销售数据、批发价格数据和商品信息
df_combined = df_sales.merge(df1, on='单品编码', how='inner')

# 合并平均损耗率数据
df_combined = df_combined.merge(df4, on='分类名称', how='inner')

# 计算销售总量
sales_total = df_combined.groupby(['日期', '分类名称'])['销量(千克)'].sum().reset_index()

# 计算平均损耗率
avg_loss_rate = df_combined.groupby(['日期', '分类名称'])['平均损耗率(%)'].mean().reset_index()

# 计算成本
cost = df_combined.groupby(['日期', '分类名称'])['批发价格(元/千克)'].mean().reset_index()

# 合并销售总量、平均损耗率和成本数据
sales_cost_loss = sales_total.merge(avg_loss_rate, on=['日期', '分类名称'], how='inner')
sales_and_cost = sales_cost_loss.merge(cost, on=['日期', '分类名称'], how='inner')

# 计算成本加成定价的加成率，考虑平均损耗率
```

```
sales_and_cost['成本加成定价的加成率'] = ((sales_and_cost['销量(千
克)'] - (sales_and_cost['销量(千克)'] * sales_and_cost['平均损耗率
(%)'] / 100)) - sales_and_cost['批发价格(元/千克)']) / sales_and_cost['批发价格
(元/千克)']

# 区分分类名称
unique_categories = sales_cost_loss['分类名称'].unique()
# 创建一个子图的布局，2 行 3 列
fig, axes = plt.subplots(2, 3, figsize=(12, 8))  # 2 行 3 列的布局

# 针对每个分类绘制散点图
for i, category in enumerate(unique_categories):
    row = i // 3  # 行索引
    col = i % 3   # 列索引

    category_data = sales_and_cost[sales_and_cost['分类名称'] == category]

    X = category_data['成本加成定价的加成率'].values.reshape(-1, 1)
    y = category_data['销量(千克)'].values

    ridge = Ridge()
    ridge.fit(X, y)

    # 预测销量
    y_pred = ridge.predict(X)

    print(f'{category}品类岭回归系数为：{ridge.coef_}')

    # 绘制散点图和拟合线
    axes[row, col].scatter(X, y, label='实际销量')
    axes[row, col].plot(X, y_pred, color='red', linewidth=2, label='岭回归拟合
线')
    axes[row, col].set_xlabel('成本加成定价的加成率')
    axes[row, col].set_ylabel('销量(千克)')
    axes[row, col].set_title(f'分类名称：{category}')
    axes[row, col].grid(True)
    axes[row, col].legend()

# 调整子图之间的间距
plt.tight_layout()
# 图中可以猜测是供需关系导致的成本加成定价和销售量的正相关关系
# 显示图形
plt.savefig('image/各品类的成本加成率与销量之间的关系图.png')
plt.show()
```

```
水生根茎类品类岭回归系数为：[6.18248218]
花叶类品类岭回归系数为：[1.89443031]
花菜类品类岭回归系数为：[4.19965997]
茄类品类岭回归系数为：[3.50344322]
```

```
辣椒类品类岭回归系数为：[2.21456039]
食用菌品类岭回归系数为：[5.80601339]


[337]
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
import numpy as np

df1 = pd.read_csv('data/附件 1.csv').iloc[:, 1:]
df2 = pd.read_csv('data/附件 2.csv').iloc[:, 1:]
df3 = pd.read_csv('data/附件 3.csv').iloc[:, 1:]
df4 = pd.read_csv('data/附件 4.csv').iloc[:, 1:]

# 合并销售数据和定价数据
merged_df = pd.merge(df2, df3, left_on=['销售日期', '单品编码'], right_on=['日期
', '单品编码'], how='inner')
# 计算每日每个品类的平均定价和平均销量
merged_df = pd.merge(merged_df, df1, how='left')
daily_avg_data = merged_df.groupby(['销售日期', '分类名称'])[['批发价格(元/千
克)', '销量(千克)']].mean().reset_index()

# 获取唯一的分类编码
category_names = daily_avg_data['分类名称'].unique()

# 创建 2x3 子图
fig, axes = plt.subplots(2, 3, figsize=(18, 12))
fig.subplots_adjust(wspace=0.3, hspace=0.3)

# 循环绘制不同品类的图形和岭回归
for i, category_name in enumerate(category_names):
    row, col = divmod(i, 3)
    ax = axes[row, col]

    # 提取当前分类数据
    category_data = daily_avg_data[daily_avg_data['分类名称'] == category_name]

    # 绘制散点图
    ax.scatter(category_data['批发价格(元/千克)'], category_data['销量(千
克)'], label=category_name)
    ax.set_xlabel('平均定价(元/千克)')
    ax.set_ylabel('平均销量(千克)')
    ax.set_title(f'分类名称: {category_name}')
    ax.grid(True)

    # 进行岭回归分析
```

```
    X = category_data['批发价格(元/千克)'].values.reshape(-1, 1)
    y = category_data['销量(千克)'].values
    ridge = Ridge(alpha=1.0)
    ridge.fit(X, y)

    # 绘制岭回归线
    x_range = np.linspace(min(X), max(X), 100).reshape(-1, 1)
    y_pred = ridge.predict(x_range)
    ax.plot(x_range, y_pred, color='red', linestyle='--', label='岭回归')

    # 计算均方误差
    mse = mean_squared_error(y, ridge.predict(X))

    ax.legend()

# 显示图形
plt.savefig('image/各分类的售价与日平均销量关系.png')
plt.show()

# 输出岭回归的系数
for i, category_name in enumerate(category_names):
    X = daily_avg_data[daily_avg_data['分类名称'] == category_name]['批发价格
(元/千克)'].values.reshape(-1, 1)
    y = daily_avg_data[daily_avg_data['分类名称'] == category_name]['销量(千
克)'].values
    ridge = Ridge(alpha=1.0)
    ridge.fit(X, y)
    print(f'分类名称: {category_name}')
    print(f'岭回归系数: {ridge.coef_}')
```

```
分类名称: 水生根茎类
岭回归系数: [-0.03417075]
分类名称: 花叶类
岭回归系数: [-0.08418141]
分类名称: 花菜类
岭回归系数: [-0.00857735]
分类名称: 茄类
岭回归系数: [-0.01247643]
分类名称: 辣椒类
岭回归系数: [-0.01914358]
分类名称: 食用菌
岭回归系数: [-0.06812304]
```

```
[550]
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
from statsmodels.tsa.arima.model import ARIMA

# 读取数据
df1 = pd.read_csv('data/附件 1.csv').iloc[:, 1:]
df2 = pd.read_csv('data/附件 2.csv').iloc[:, 1:]
df3 = pd.read_csv('data/附件 3.csv').iloc[:, 1:]
df4 = pd.read_csv('data/附件 4.csv').iloc[:, 1:]

# 合并 df1 和 df2，匹配商品信息和销售数据
merged_data = pd.merge(df2, df1, on='单品编码', how='inner')

# 将销售日期列设置为日期时间类型
merged_data['销售日期'] = pd.to_datetime(merged_data['销售日期'])

# 按分类编码和销售日期分组，计算每日销售总量
daily_sales_by_category = merged_data.groupby(['分类编码', pd.Grouper(key='销售
日期', freq='D')])['销量(千克)'].sum().reset_index()

# 过滤出要预测的日期范围
start_date = '2023-07-01'
end_date = '2023-07-07'
fore-
cast_days = (pd.to_datetime(end_date) - pd.to_datetime(start_date)).days + 1

# 创建一个空的 DataFrame 用于存储预测结果
forecast_result = pd.DataFrame(columns=['分类编码', '销售日期', '销售量(千克)'])

# 遍历每个分类编码，使用 ARIMA 模型预测销售量
for category_code, group_data in daily_sales_by_category.groupby('分类编码'):
    sales_data = group_data[group_data['销售日期'] < start_date]['销量(千克)']

    # 拟合 ARIMA 模型
    p, d, q = 1, 1, 1  # 选择合适的 ARIMA 模型阶数
    model = ARIMA(sales_data, order=(p, d, q))
    model_fit = model.fit()

    # 预测未来日期的销售量
    forecast = model_fit.forecast(steps=forecast_days)

    # 创建一个 DataFrame 存储预测结果
    forecast_df = pd.DataFrame({
        '分类编码': [category_code] * forecast_days,
        '销售日期': pd.date_range(start=start_date, end=end_date),
        '销售量(千克)': forecast
    })

    # 将预测结果添加到总结果中
    forecast_result = forecast_result.append(forecast_df, ignore_index=True)
```

```
# 打印预测结果
print("2023 年 7 月 1 日至 7 月 7 日的销售量预测：")
forecast_result = pd.merge(forecast_result, df1[['分类编码', '分类名称']], on='
分类编码', how='left')
forecast_result = forecast_result.drop_duplicates(subset=['分类编码','销售日期
']).reset_index(drop=True)
forecast_result
```

```
2023 年 7 月 1 日至 7 月 7 日的销售量预测：
```

```
[551]
# 合并 df3 和 df1，匹配商品信息和批发价格数据
merged_price_data = pd.merge(df3, df1, on='单品编码', how='inner')

# 将日期列设置为日期时间类型
merged_price_data['日期'] = pd.to_datetime(merged_price_data['日期'])

# 按分类编码和日期分组，计算每日平均批发价格
daily_avg_price_by_category = merged_price_data.groupby(['分类编码
', pd.Grouper(key='日期', freq='D')])['批发价格(元/千克)'].mean().reset_index()

# 创建一个空的 DataFrame 用于存储批发价格预测结果
price_forecast_result = pd.DataFrame(columns=['分类编码', '销售日期', '平均批发
价格(元/千克)'])

# 遍历每个分类编码，使用 ARIMA 模型预测平均批发价格
for category_code, group_data in daily_avg_price_by_category.groupby('分类编码
'):
    price_data = group_data[group_data['日期'] < start_date]['批发价格(元/千
克)']

    # 拟合 ARIMA 模型
    p, d, q = 1, 1, 1  # 选择合适的 ARIMA 模型阶数
    model = ARIMA(price_data, order=(p, d, q))
    model_fit = model.fit()

    # 预测未来日期的平均批发价格
    price_forecast = model_fit.forecast(steps=forecast_days)

    # 创建一个 DataFrame 存储批发价格预测结果
    price_forecast_df = pd.DataFrame({
        '分类编码': [category_code] * forecast_days,
        '销售日期': pd.date_range(start=start_date, end=end_date),
        '平均批发价格(元/千克)': price_forecast
    })

    # 将批发价格预测结果添加到总结果中
```

```
    price_forecast_result = price_forecast_result.append(price_fore-
cast_df, ignore_index=True)

# 打印批发价格预测结果
print("2023 年 7 月 1 日至 7 月 7 日的平均批发价格预测：")
price_forecast_result = pd.merge(price_forecast_result, df1[['分类编码', '分类
名称']], on='分类编码', how='left')
price_forecast_result = price_forecast_result.drop_duplicates(subset=['分类编码
','销售日期']).reset_index(drop=True)
price_forecast_result
```

```
2023 年 7 月 1 日至 7 月 7 日的平均批发价格预测：
```

```
[552]
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA

# 读取数据
df1 = pd.read_csv('data/附件 1.csv').iloc[:, 1:]
df2 = pd.read_csv('data/附件 2.csv').iloc[:, 1:]
df3 = pd.read_csv('data/附件 3.csv').iloc[:, 1:]
df4 = pd.read_csv('data/附件 4.csv').iloc[:, 1:]

# 合并 df1 和 df2，匹配商品信息和销售数据
merged_data = pd.merge(df2, df1, on='单品编码', how='inner')

# 将销售日期列设置为日期时间类型
merged_data['销售日期'] = pd.to_datetime(merged_data['销售日期'])

# 按分类编码和销售日期分组，计算每日销售单价(元/千克)的平均值
daily_price_by_category = merged_data.groupby(['分类编码', pd.Grouper(key='销售
日期', freq='D')])['销售单价(元/千克)'].mean().reset_index()

# 过滤出要预测的日期范围
start_date = '2023-07-01'
end_date = '2023-07-07'
fore-
cast_days = (pd.to_datetime(end_date) - pd.to_datetime(start_date)).days + 1

# 创建一个空的 DataFrame 用于存储预测结果
forecast_Price = pd.DataFrame(columns=['分类编码', '销售日期', '销售单价(元/千
克)'])

# 遍历每个分类编码，使用 ARIMA 模型预测销售单价(元/千克)
for category_code, group_data in daily_price_by_category.groupby('分类编码'):
    price_data = group_data[group_data['销售日期'] < start_date]['销售单价(元/千
克)']
```

```
    # 拟合 ARIMA 模型
    p, d, q = 1, 1, 1  # 选择合适的 ARIMA 模型阶数
    model = ARIMA(price_data, order=(p, d, q))
    model_fit = model.fit()

    # 预测未来日期的销售单价(元/千克)
    forecast = model_fit.forecast(steps=forecast_days)

    # 创建一个 DataFrame 存储预测结果
    forecast_df = pd.DataFrame({
        '分类编码': [category_code] * forecast_days,
        '销售日期': pd.date_range(start=start_date, end=end_date),
        '销售单价(元/千克)': forecast
    })

    # 将预测结果添加到总结果中
    forecast_Price = forecast_Price.append(forecast_df, ignore_index=True)

# 打印预测结果
print("2023 年 7 月 1 日至 7 月 7 日的销售单价(元/千克)预测：")
forecast_Price = pd.merge(forecast_Price, df1[['分类编码', '分类名称']], on='分类编码', how='left')
forecast_Price = forecast_Price.drop_duplicates(subset=['分类编码','销售日期']).reset_index(drop=True)
forecast_Price
```

2023 年 7 月 1 日至 7 月 7 日的销售单价(元/千克)预测：

```
[557]
merged_result = pd.merge(forecast_result, price_forecast_result, on=['销售日期', '分类编码','分类名称'], how='left')
df4 = df4.rename(columns={'小分类编码': '分类编码', '小分类名称': '分类名称', '平均损耗率(%)_小分类编码_不同值': '平均损耗率(%)'})
merged_result = pd.merge(merged_result, forecast_Price, on=['销售日期', '分类编码','分类名称'], how='left')
merged_result = pd.merge(merged_result, df4, on=['分类编码','分类名称'], how='left')

merged_result
[654]
import pandas as pd

df1 = pd.read_csv('data/附件 1.csv').iloc[:, 1:]
df2 = pd.read_csv('data/附件 2.csv').iloc[:, 1:]

# 将销售日期列转换为日期时间格式
df2['销售日期'] = pd.to_datetime(df2['销售日期'])
# 过滤出销售类型为"销售"且是否打折销售为"否"的数据
```

```
filtered_df2 = df2[(df2['销售类型'] == '销售') & (df2['是否打折销售'] == '否')]
# 合并 df1 和过滤后的 df2 以获取每个商品的分类信息
merged_df = pd.merge(filtered_df2, df1, on='单品编码', how='inner')
# 根据分类、销售日期和单品编码进行分组，并计算每日销量和每日销售总额
result = merged_df.groupby(['分类名称', '销售日期', '单品编码'])[['销量(千克)', '
销售单价(元/千克)']].agg({'销量(千克)': 'sum', '销售单价(元/千
克)': 'mean'}).reset_index()
# 计算每日销售总额
result['销售总额'] = result['销量(千克)'] * result['销售单价(元/千克)']
# 根据分类和销售日期进行分组，并计算每日销量和平均销售价格
final_result = result.groupby(['分类名称', '销售日期'])[['销量(千克)', '销售总额
']].agg({'销量(千克)': 'sum', '销售总额': 'sum'}).reset_index()
final_result['平均销售价格'] = final_result['销售总额'] / final_result['销量(千
克)']
final_result = final_result.merge(df4,on=['分类名称'],how='left')
final_result
[538]
df6 = final_result.copy()

# 循环处理每个品类
unique_categories = df6['分类名称'].unique()
elasticities = []

for category_code in unique_categories:
    category_data = df6[df6['分类名称'] == category_code]
    X = category_data['平均销售价格']
    X = sm.add_constant(X)  # 添加截距项
    y = category_data['销量(千克)']

    # 拟合线性回归模型
    model = sm.OLS(y, X).fit()

    # 提取价格弹性系数 (β1)
    elasticity = model.params['平均销售价格']
    elasticities.append(elasticity)

# 创建一个 DataFrame 来存储每个品类的价格弹性系数
elasticities_df = pd.DataFrame({'分类名称': unique_categories, '价格弹性系数
': elasticities})

# 输出价格弹性系数
elasticities_df
[606]
merged_result
[601]
import pandas as pd
import numpy as np
from scipy.optimize import minimize
from scipy.optimize import Bounds
```

```
df5 = merged_result.copy()

# 定义计算成本的函数
def calculate_cost(sales, loss_rate, wholesale_price):
    cost = sales / (1 - loss_rate * 0.01) * wholesale_price
    return cost

# 定义计算收益的函数
def calculate_profit(cost, markup_rate):
    profit = cost * markup_rate
    return profit

# 定义弹性需求模型函数
def elastic_demand(price_elasticity, initial_price, ini-
tial_sales, cost, markup_rate):
    def objective(x):
        new_price, new_sales = x
        new_cost = calculate_cost(new_sales, loss_rate, wholesale_price)
        new_profit = calculate_profit(new_cost, markup_rate)
        return -new_profit  # 目标函数是最大化收益

    bounds = Bounds([initial_price * 0.5, initial_sales * 0.5], [ini-
tial_price * 2, initial_sales * 2])

    constraints = [
        {'type': 'eq', 'fun': lambda x: x[0] - (calcu-
late_cost(x[1], loss_rate, wholesale_price) + calcu-
late_cost(x[1], loss_rate, wholesale_price) * markup_rate)},
        {'type': 'ineq', 'fun': lambda x: x[1] - initial_sales * 0.9},  # 不低
于10%的限制
        {'type': 'ineq', 'fun': lambda x: initial_sales * 1.1 - x[1]},  # 不高
于10%的限制
        {'type': 'ineq', 'fun': lambda x: x[0] - initial_price * 0.9},  # 不低
于10%的售价
        {'type': 'ineq', 'fun': lambda x: initial_price * 1.1 - x[0]}  # 不高
于10%的售价
    ]

    result = minimize(objective, [initial_price, ini-
tial_sales], bounds=bounds, constraints=constraints)
    new_price, new_sales = result.x
    return new_price, new_sales

# 循环处理每个品类
unique_categories = df5['分类名称'].unique()
optimized_prices = []
optimized_sales = []
```

```
for category_code in unique_categories:
    category_data = df5[df5['分类名称'] == category_code]
    loss_rate = category_data['平均损耗率(%)'].iloc[0]
    wholesale_price = category_data['平均批发价格(元/千克)'].iloc[0]
    price_elasticity = elasticities_df[elasticities_df['分类名称'] == cate-
gory_code]['价格弹性系数'].iloc[0]

    # 获取品类的所有销售日期
    sale_dates = category_data['销售日期'].unique()

    # 按照日期进行处理
    for sale_date in sale_dates:
        # 获取当日的数据
        daily_data = category_data[category_data['销售日期'] == sale_date]
        initial_price = daily_data['销售单价(元/千克)'].iloc[0]
        initial_sales = daily_data['销售量(千克)'].iloc[0]

        # 使用弹性需求模型优化售价和销量
        new_price, new_sales = elastic_demand(price_elasticity, ini-
tial_price, initial_sales, loss_rate, wholesale_price)

        # 将优化后的售价和销量添加到列表中
        optimized_prices.append(new_price)
        optimized_sales.append(new_sales)

# 将优化后的售价和销量保存回 df5
df5['新售价'] = optimized_prices
df5['新销量'] = optimized_sales
[602]
# 计算新的补货量
df5['新补货量'] = df5.apply(lambda row: row['新销量'] / (1 - row['平均损耗率
(%)']*0.01), axis=1)

# 定义计算成本的函数
def calculate_cost(row):
    replenishment = row['新销量']
    loss_rate = row['平均损耗率(%)']
    wholesale_price = row['平均批发价格(元/千克)']
    return replenishment / (1 - (loss_rate / 100)) * wholesale_price

# 使用 apply 方法计算每一行的成本
df5['新成本'] = df5.apply(calculate_cost, axis=1)
[603]
df5['新收益']= df5['新售价']*df5['新销量'] - df5['新成本']
df5.groupby(['分类名称'])['新收益'].sum()
分类名称
水生根茎类      372.807240
```

```
花叶类       1142.924503
花菜类        432.106383
茄类         638.416979
辣椒类        761.516707
食用菌       2435.705169
Name: 新收益, dtype: float64
[604]
df5
[611]
df6
```

第三问代码

```
[804]
import pandas as pd

df1 = pd.read_csv('data/附件 1.csv').iloc[:, 1:]
df2 = pd.read_csv('data/附件 2.csv').iloc[:, 1:]

# 将销售日期列转换为日期时间格式
df2['销售日期'] = pd.to_datetime(df2['销售日期'])
# 过滤出销售类型为"销售"且是否打折销售为"否"的数据
filtered_df2 = df2[(df2['销售类型'] == '销售') & (df2['是否打折销售'] == '否')]
# 合并 df1 和过滤后的 df2 以获取每个商品的分类信息
merged_df = pd.merge(filtered_df2, df1, on='单品编码', how='inner')
# 根据分类、销售日期、单品编码进行分组，并计算每日销量和每日销售总额
result = merged_df.groupby(['分类名称', '销售日期', '单品编码'])[['销量(千克)', '
销售单价(元/千克)']].agg({'销量(千克)': 'sum', '销售单价(元/千
克)': 'mean'}).reset_index()
# 计算每日销售总额
result['销售总额'] = result['销量(千克)'] * result['销售单价(元/千克)']

# 根据分类、销售日期分组，并计算每个单品的销量、销售总额和平均销售价格
individual_product_result = result.groupby(['分类名称', '销售日期', '单品编码
'])[['销量(千克)', '销售总额']].agg({'销量(千克)': 'sum', '销售总额': 'sum'}).re-
set_index()
individual_product_result['平均销售价格'] = individual_product_result['销售总额
'] / individual_product_result['销量(千克)']

individual_product_result = individual_product_result.merge(df1, on='单品编码
', how='left')
individual_product_result = individual_product_result[['分类名称_x', '销售日期
', '单品编码', '销量(千克)', '销售总额', '平均销售价格', '单品名称', '分类编码']]
individual_product_result.rename(columns={'分类名称_x': '分类名称
'}, inplace=True)

# 将销售日期列转换为日期时间格式
individual_product_result['销售日期'] = pd.to_datetime(individual_product_re-
sult['销售日期'])
```

```
# 设置筛选条件
start_date = '2023-06-24'
end_date = '2023-06-30'

# 使用条件筛选选择符合条件的行
filtered_result = individual_product_result[(individual_product_result['销售日
期'] >= start_date) & (individual_product_result['销售日期'] <= end_date)]
filtered_result
[798]
df1 = pd.read_csv('data/附件 1.csv').iloc[:, 1:]
df2 = pd.read_csv('data/附件 2.csv').iloc[:, 1:]

# 将销售日期列转换为日期时间格式
df2['销售日期'] = pd.to_datetime(df2['销售日期'])
# 过滤出销售类型为"销售"且是否打折销售为"否"的数据
filtered_df2 = df2[(df2['销售类型'] == '销售') & (df2['是否打折销售'] == '否')]
# 合并 df1 和过滤后的 df2 以获取每个商品的分类信息
merged_df = pd.merge(filtered_df2, df1, on='单品编码', how='inner')
# 根据分类、销售日期、单品编码进行分组，并计算每日销量和每日销售总额
result = merged_df.groupby(['分类名称', '销售日期', '单品编码'])[['销量(千克)', '
销售单价(元/千克)']].agg({'销量(千克)': 'sum', '销售单价(元/千
克)': 'mean'}).reset_index()
# 计算每日销售总额
result['销售总额'] = result['销量(千克)'] * result['销售单价(元/千克)']

# 根据分类、销售日期分组，并计算每个单品的销量、销售总额和平均销售价格
individual_product_result = result.groupby(['分类名称', '销售日期', '单品编码
'])[['销量(千克)', '销售总额']].agg({'销量(千克)': 'sum', '销售总额': 'sum'}).re-
set_index()
individual_product_result['平均销售价格'] = individual_product_result['销售总额
'] / individual_product_result['销量(千克)']

individual_product_result = individual_product_result.merge(df1, on='单品编码
', how='left')
individual_product_result = individual_product_result[['分类名称_x', '销售日期
', '单品编码', '销量(千克)', '销售总额', '平均销售价格', '单品名称', '分类编码']]
individual_product_result.rename(columns={'分类名称_x': '分类名称
'}, inplace=True)

# 将销售日期列转换为日期时间格式
individual_product_result['销售日期'] = pd.to_datetime(individual_product_re-
sult['销售日期'])

filtered_df = individual_product_result[individual_product_result['单品名称
'].isin(filtered_result['单品名称'].unique())]
filtered_df
[687]
```

```
df7 = filtered_df.copy()

# 循环处理每个品类
unique_categories = df7['单品名称'].unique()
elasticities = []

for category_code in unique_categories:
    category_data = df7[df7['单品名称'] == category_code]
    X = category_data['平均销售价格']
    X = sm.add_constant(X)  # 添加截距项
    y = category_data['销量(千克)']

    # 拟合线性回归模型
    model = sm.OLS(y, X).fit()

    # 提取价格弹性系数 (β1)
    elasticity = model.params['平均销售价格']
    elasticities.append(elasticity)

# 创建一个DataFrame来存储每个品类的价格弹性系数
elasticities_df = pd.DataFrame({'单品名称': unique_categories, '价格弹性系数
': elasticities})

# 价格弹性系数都很低，不合适之前的做法了

# 输出价格弹性系数
elasticities_df
[940]
q3df = filtered_df.copy()
q3df = q3df.rename(columns={'销售日期':'日期'})
q3df['日期'] = q3df['日期'].astype('str')
q3df = q3df.merge(df3,on=['单品编码','日期'],how='left')
q3df = q3df.merge(df4,on=['分类名称','分类编码'],how='left')
[941]
q3df['收益'] = (q3df['销量(千克)']*q3df['平均销售价格'])-(q3df['销量(千
克)']/(1 - q3df['平均损耗率(%)']))*q3df['批发价格(元/千克)']
[942]
required_fields = ['单品名称', '日期', '销量(千克)','平均销售价格', '批发价格(元/
千克)', '平均损耗率(%)', '收益']
q3df = q3df[required_fields]
q3df
[974]
import pandas as pd
import numpy as np
from statsmodels.tsa.arima.model import ARIMA

# 设置ARIMA模型的阶数
p = 1  # 自回归阶数
d = 1  # 差分阶数
```

```
q = 1  # 移动平均阶数

# 创建一个空的DataFrame，用于存储每个单品的预测结果
forecast_results = pd.DataFrame(columns=['单品名称', '日期', '销量预测', '平均销售价格预测', '批发价格预测'])
unique_products = q3df['单品名称'].unique()

# 遍历每个单品并执行ARIMA 预测
for product_name in unique_products:
    # 获取单品的历史销量时间序列
    sales_series = q3df[q3df['单品名称'] == product_name].set_index('日期')['销量(千克)']

    # 拟合ARIMA 模型
    model = ARIMA(sales_series, order=(p, d, q))
    model_fit = model.fit()

    # 预测2023 年7 月1 日的销量
    forecast_sales = model_fit.forecast(steps=1).iloc[0]  # 直接提取预测值

    # 拟合ARIMA 模型来预测平均销售价格
    avg_price_series = q3df[q3df['单品名称'] == product_name].set_index('日期')['平均销售价格']
    model_avg_price = ARIMA(avg_price_series, order=(p, d, q))
    model_avg_price_fit = model_avg_price.fit()
    forecast_avg_price = model_avg_price_fit.forecast(steps=1).iloc[0]  # 直接提取预测值

    # 拟合ARIMA 模型来预测批发价格
    wholesale_price_series = q3df[q3df['单品名称'] == product_name].set_index('日期')['批发价格(元/千克)']
    model_wholesale_price = ARIMA(wholesale_price_series, order=(p, d, q))
    model_wholesale_price_fit = model_wholesale_price.fit()
    forecast_wholesale_price = model_wholesale_price_fit.forecast(steps=1).iloc[0]  # 直接提取预测值

    # 将预测结果添加到结果DataFrame 中
    forecast_results = forecast_results.append({
        '单品名称': product_name,
        '日期': '2023-07-01',
        '销量预测': forecast_sales,
        '平均销售价格预测': forecast_avg_price,
        '批发价格预测': forecast_wholesale_price
    }, ignore_index=True)

forecast_results = forecast_results.merge(df1,on='单品名称',how='left')
forecast_results = forecast_results.merge(df4,on='分类名称',how='left')
forecast_results = forecast_results.drop(['分类编码_x','分类编码_y','单品编码
```

```
'],axis=1)
forecast_results
[1040]
import random
import numpy as np
import pandas as pd
from deap import base, creator, tools, algorithms
import matplotlib.pyplot as plt

# 定义遗传算法的相关参数
POP_SIZE = 50  # 种群大小
GENE_LENGTH = len(forecast_results)
NUM_GENERATIONS = 100  # 迭代代数
CXPB = 0.7  # 交叉概率
MUTPB = 0.2  # 变异概率

# 创建适应度函数
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

# 初始化种群
toolbox = base.Toolbox()
toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individ-
ual, toolbox.attr_bool, n=GENE_LENGTH)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)

# 定义适应度函数
def evaluate(individual):
    # 计算每个品类的销量、销售价格和批发价格
    selected_categories = [forecast_re-
sults.iloc[i] for i in range(GENE_LENGTH) if individual[i] == 1]
    total_revenue = 0
    total_sales = 0

    for category in selected_categories:
        # 进行销售收益的计算，使用正确的收益计算方式
        sales = category['销量预测']
        avg_price = category['平均销售价格预测']

        if sales < 2.5:
            sales = 2.5

        total_sales += sales
        total_revenue += (sales * avg_price) - (category['销量预测
```

```
'] / (1 - category['平均损耗率(%)'] / 100)) * category['批发价格预测']

    # 添加约束条件：单品数限制在 27-33 之间
    if len(selected_categories) < 27 or len(selected_categories) > 33:
        return -1e9,  # 为不满足约束的个体分配极低的适应度

    return total_revenue,

toolbox.register("evaluate", evaluate)

# 创建种群
pop = toolbox.population(n=POP_SIZE)

# 创建一个空列表来存储每一代的最佳适应度值
best_fitness_values = []

# 迭代优化
for gen in range(NUM_GENERATIONS):
    # 选择、交叉和变异操作
    offspring = algorithms.varAnd(pop, toolbox, cxpb=CXPB, mutpb=MUTPB)

    # 评估适应度
    fitnesses = list(map(toolbox.evaluate, offspring))
    for ind, fit in zip(offspring, fitnesses):
        ind.fitness.values = fit

    # 选择下一代种群
    pop = toolbox.select(offspring, k=len(pop))

    # 获取最佳个体的适应度值并添加到列表中
    best_ind = tools.selBest(pop, k=1)[0]
    best_fitness = best_ind.fitness.values[0]
    best_fitness_values.append(best_fitness)

# 绘制迭代曲线
plt.figure()
plt.plot(range(1, NUM_GENERATIONS + 1), best_fitness_values, linestyle='-')
plt.xlabel('迭代次数')
plt.ylabel('最佳售价')
plt.savefig('image/迭代次数与最佳利润关系图.png')
plt.grid(True)
plt.show()

# 获取最佳个体
best_individual = tools.selBest(pop, k=1)[0]
selected_categories = [forecast_re-
sults.iloc[i] for i in range(GENE_LENGTH) if best_individual[i] == 1]
```

```
# 输出最佳组合和总收益
print("最佳组合：")
for category in selected_categories:
    print(category['单品名称'])
print("总收益：", evaluate(best_individual)[0])
```

```
最佳组合:
高瓜(1)
洪湖藕带
净藕(1)
红莲藕带
菱角
野生粉藕
高瓜(2)
云南生菜
竹叶菜
上海青
菠菜
娃娃菜
外地茼蒿
奶白菜
云南油麦菜(份)
菠菜(份)
云南生菜(份)
西兰花
枝江青梗散花
紫茄子(2)
长线茄
螺丝椒
芜湖青椒(1)
小米椒(份)
小皱皮(份)
螺丝椒(份)
七彩椒(2)
姜蒜小米椒组合装(小份)
红椒(2)
白玉菇(袋)
西峡花菇(1)
金针菇(盒)
双孢菇(盒)
总收益：  755.4182839807803
```

```
[1034]
selected_categoriesList = [x['单品名称'] for x in selected_categories]
[1035]
q3df2 = forecast_results[forecast_results['单品名称'].isin(selected_categories-
List)]
[1036]
q3df2['销量预测'] = q3df2['销量预测'].apply(lambda x: 2.5 if x < 2.5 else x)
```

```
[1037]
q3df2['收益'] =  q3df2['销量预测']*q3df2['平均销售价格预测'] - (q3df2['销量预测
']/(1 - q3df2['平均损耗率(%)'] / 100))*q3df2['批发价格预测']
[1038]
q3df2['补货量'] = q3df2['销量预测']/((1 - q3df2['平均损耗率(%)']) / 100)
[1039]
q3df2[['单品名称','补货量','平均销售价格预测','收益']]
```

	单品名称	补货量	平均销售价格预测	收益
0	高瓜(1)	-19.76284585	16	6.209625088
1	洪湖藕带	-21.93900152	25.89143089	13.97612659
2	净藕(1)	-37.63475185	16	19.05476639
3	红莲藕带	-19.76284585	9.2	7.681270038
4	菱角	-19.76284585	14.00077014	8.482244344
5	野生粉藕	-19.76284585	26	17.82877046
6	高瓜(2)	-19.76284585	17.99916501	6.41640842
9	竹叶菜	-116.0787863	3.399709426	12.83739514
10	上海青	-43.12007888	8	16.87281501
12	木耳菜	-37.1532317	6.02375418	10.39303391
14	菠菜	-53.68460069	13.99664466	19.0957451
15	娃娃菜	-80.70586668	6.8	16.26507667
18	奶白菜	-86.31236984	3.947509585	10.1769114
19	小青菜(1)	-50.55883258	5.207361044	12.63383262
20	云南油麦菜(份)	-154.5100022	4.5	22.67875392
21	菠菜(份)	-83.69281684	6.810863706	21.13360142
22	云南生菜(份)	-263.8170067	5.883501576	56.36800525
23	木耳菜(份)	-21.13271344	3.9	4.887270968
24	西兰花	-105.0412331	11.23960423	34.3829547
25	枝江青梗散花	-44.10082139	12.90536102	19.38734044
26	紫茄子(2)	-252.7174189	6.000016265	33.73303938
29	长线茄	-102.6592167	11.99513385	26.2173585
30	紫茄子(1)	-47.40238106	9.000041033	11.08888583
31	螺丝椒	-93.67086255	11.98955401	16.53547608
32	芜湖青椒(1)	-201.2116866	5.202394822	19.69608913
33	小米椒(份)	-263.2678519	5.800122482	75.75073752
36	螺丝椒(份)	-133.6072263	5.902124161	17.64399032
37	七彩椒(2)	-30.33980583	20.86979371	18.46588766
38	姜蒜小米椒组合装(小份)	-90.91570483	4.8	16.57798724
40	红椒(2)	-30.33980583	20	14.95166207
42	西峡花菇(1)	-72.43583495	23.88611001	41.7806356
44	金针菇(盒)	-173.8728943	2.044275636	6.49439152
46	双孢菇(盒)	-104.9914755	5.5	15.4263388

第四问代码

```
[1041]
```

```
q1_df
[1072]
import numpy as np
from scipy.linalg import eig

# 创建两两比较矩阵
matrices = {
    "天气状况": np.array([[1, 3, 2],
                      [1/3, 1, 1/2],
                      [1/2, 2, 1]]),

    "节假日": np.array([[1, 3, 2],
                     [1/3, 1, 1/2],
                     [1/2, 2, 1]]),

    "销售趋势": np.array([[1, 1/3, 3],
                       [3, 1, 4],
                       [1/3, 1/4, 1]]),

    "季节性需求": np.array([[1, 3, 2, 2],
                         [1/3, 1, 1/2, 1/2],
                         [1/2, 2, 1, 1],
                         [1/2, 2, 1, 1]]),

    "市场竞争": np.array([[1, 1/3, 1/4],
                      [3, 1, 1/2],
                      [4, 2, 1]]),

    "供应链可靠性": np.array([[1, 2, 4],
                           [1/2, 1, 2],
                           [1/4, 1/2, 1]])
}

# 计算每个矩阵的特征向量和特征值
eigenvalues = {}
eigenvectors = {}
for criterion, matrix in matrices.items():
    w, v = eig(matrix)
    eigenvalues[criterion] = max(w.real)  # 取实部最大的特征值
    eigenvectors[criterion] = v[:, np.argmax(w.real)]  # 对应的特征向量

# 计算权重
weights = {}
for criterion, eigval in eigenvalues.items():
    weights[criterion] = eigval / sum(eigenvalues.values())

# 输出结果
print("各因素的权重:")
```

```python
for criterion, weight in weights.items():
    print(f"{criterion}: {weight}")

# 计算子准则的权重
subcriteria_weights = {}
subcriteria_matrices = {
    "天气状况": np.array([[1, 3, 2],
                        [1/3, 1, 1/2],
                        [1/2, 2, 1]]),

    "节假日": np.array([[1, 3, 2],
                       [1/3, 1, 1/2],
                       [1/2, 2, 1]]),

    "销售趋势": np.array([[1, 1/3, 3],
                         [3, 1, 4],
                         [1/3, 1/4, 1]]),

    "季节性需求": np.array([[1, 3, 2, 2],
                           [1/3, 1, 1/2, 1/2],
                           [1/2, 2, 1, 1],
                           [1/2, 2, 1, 1]]),

    "市场竞争": np.array([[1, 1/3, 1/4],
                         [3, 1, 1/2],
                         [4, 2, 1]]),

    "供应链可靠性": np.array([[1, 2, 4],
                             [1/2, 1, 2],
                             [1/4, 1/2, 1]])
}

for subcriterion, matrix in subcriteria_matrices.items():
    w, v = eig(matrix)
    subcriteria_weights[subcriterion] = w.real / sum(w.real)

print("\n各子准则的权重:")
for subcriterion, weight in subcriteria_weights.items():
    print(f"{subcriterion}: {weight}")
```

```
各因素的权重:
天气状况: 0.1573803333389768
节假日: 0.1573803333389768
销售趋势: 0.16074376814732552
季节性需求: 0.20974068901979795
市场竞争: 0.15785584172893635
供应链可靠性: 0.15689903442598654
```

```
各子准则的权重:
天气状况: [ 1.00306757 -0.00153379 -0.00153379]
节假日: [ 1.00306757 -0.00153379 -0.00153379]
销售趋势: [ 1.02450451 -0.01225225 -0.01225225]
季节性需求: [ 1.00259073e+00 -1.29536278e-03 -1.29536278e-03 -5.99381436e-34]
市场竞争: [ 1.00609824 -0.00304912 -0.00304912]
供应链可靠性: [-7.40148683e-17  1.00000000e+00  0.00000000e+00]
```

```
[1080]
import matplotlib.pyplot as plt

# 存储权重数据
criteria_weights = {
    "天气状况": 0.1573803333389768,
    "节假日": 0.1573803333389768,
    "销售趋势": 0.16074376814732552,
    "季节性需求": 0.20974068901979795,
    "市场竞争": 0.15785584172893635,
    "供应链可靠性": 0.15689903442598654
}

subcriteria_weights = {
    "天气状况": [1.00306757, -0.00153379, -0.00153379],
    "节假日": [1.00306757, -0.00153379, -0.00153379],
    "销售趋势": [1.02450451, -0.01225225, -0.01225225],
    "季节性需求": [1.00259073, -1.29536278e-03, -1.29536278e-03, -5.99381436e-
34],
    "市场竞争": [1.00609824, -0.00304912, -0.00304912],
    "供应链可靠性": [-7.40148683e-17, 1.00000000e+00, 0.00000000e+00]
}
```

```
[1085]
# 绘制主要准则的柱状图
plt.figure(figsize=(10, 6))
plt.bar(criteria_weights.keys(), criteria_weights.values())
plt.title("主要准则的权重")
plt.xlabel("准则")
plt.ylabel("权重")
plt.savefig('image/主要准则的权重')
plt.xticks(rotation=45)
plt.show()
```